

AFRL-SN-WP-TR-2000-1046

**SOFTWARE SCIENCES AND ENGINEERING
RESEARCH**



**J. MORRIS
M. SHAW**

**CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
5000 FORBES AVENUE
PITTSBURGH, PA 15213-3891**

MAY 2000

FINAL REPORT FOR 09/30/1993 – 12/31/1999

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

**SENSORS DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE OH 45433-7318**

DTIC QUALITY INSPECTED 4

20000802 215

NOTICE

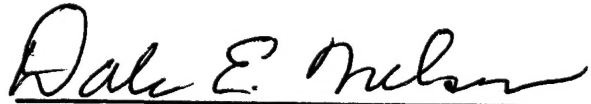
Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.



Chahira M. Hopper
Target Recognition Branch
Sensor ATR Technology Division



Dale E Nelson, Chief
Target Recognition Branch
Sensor ATR Technology Division



Jerry L Covert, Chief
Sensor ATR Technology Division
Sensor Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE MAY 2000		3. REPORT TYPE AND DATES COVERED FINAL REPORT FOR 09/30/1993 - 12/31/1999
4. TITLE AND SUBTITLE SOFTWARE SCIENCES AND ENGINEERING RESEARCH			5. FUNDING NUMBERS C F33615-93-1-1330 PE 61101 PR A724 TA 01 WU 01	
6. AUTHOR(S) J. MORRIS M. SHAW				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) CARNEGIE MELLON UNIVERSITY COMPUTER SCIENCE DEPARTMENT 5000 FORBES AVENUE PITTSBURGH, PA 15213-3891			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) SENSORS DIRECTORATE AIR FORCE RESEARCH LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT-PATTERSON AFB, OH 45433-7318 POC: CHAHIRA M. HOPPER, AFRL/SNWR, 937-255-5579 EXT. 4248			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-SN-WP-TR-2000-1046	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) During the period 30 September 1993 to 31 December 1999, Carnegie Mellon University pursued a broad set of research initiatives in response to DARPA's BAA 93-11. These projects constituted a wide-ranging research program that significantly advanced the state of the art in software engineering science and technology. The projects addressed two fundamental issues in software design and implementation, namely: <ol style="list-style-type: none"> 1. How to ensure that computing systems have the right functionality to meet user's needs. 2. How to make a computer system highly reliable and maintainable over its lifecycle. <p>All the contractual tasks attacked one or both of these questions in a different way. In 1997 under the additional task of productizing research results, speech-understanding technology was applied to multilingual interaction (including translation) and automatic meeting transcription. A comprehensive review of each effort, along with a bibliography that includes selected publication abstracts comprise the report.</p>				
14. SUBJECT TERMS software engineering, team-centered, composable, software architectures, machine learning, speech recognition, specification, verification, spoken language, foreign language, interactive communication, multimodal interaction.			15. NUMBER OF PAGES 206	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR	

Table of Contents

Introduction	1
1. Team-centered System Development	1-1
1.1 User-centered design of group planning tools	1-1
1.1.1 Preliminary activities	1-1
1.1.2 Informal interviews	1-2
1.1.3 System goals	1-3
1.2 The prototype	1-3
1.3 Research questions	1-4
1.4 Prototype development and testing	1-4
1.5 Bibliography	1-5
2. Composable Software Systems	2-1
2.1 Scientific foundation	2-1
2.1.1 In search of standard software architecture problems	2-1
2.1.2 Software development education	2-2
2.1.3 Developing a formal basis for architectural style	2-2
2.1.4 Formal specification of AEC product models	2-2
2.1.5 Decomposing and recomposing transactional concepts	2-3
2.1.6 Semantics for parallel architectures	2-3
2.1.7 Coping with heterogeneity in software architecture	2-3
2.1.8 A software design paradigm based on process control	2-3
2.1.9 A software systems architectures course	2-3
2.1.10 Architectural mismatch	2-4
2.1.11 Using architectural style	2-4
2.1.12 Subtyping in object-oriented languages	2-4
2.1.13 Composing transactional concepts	2-4
2.1.14 Applying formal methods to distributed systems	2-5
2.1.15 Automated analysis of design	2-5
2.1.16 Specifying weak sets	2-5
2.1.17 Abstract models of memory management	2-6
2.1.18 Polymorphic closure conversion	2-6
2.1.19 An architecture for a family of instrumentation systems	2-6
2.1.20 View structuring	2-7
2.1.21 Architectural issues in software reuse	2-7
2.1.22 Teaching mathematics to software engineers	2-7
2.2 Architectural languages	2-7
2.2.1 A theory and description language for component interactions	2-8
2.2.2 A constraint-based, object-oriented, programming language	2-8
2.2.3 Patterns for software architectures	2-8
2.2.4 Formalizing architectural style	2-9
2.2.5 Re-design of UniCon	2-9
2.3 Legacy systems	2-9
2.3.1 Aspect: detecting bugs with abstract dependences	2-9
2.3.2 Viewpoint composition in model-based specifications	2-10
2.3.3 Reverse engineering	2-10
2.3.4 Advances in reverse engineering	2-10
2.4 Tools and environments	2-11
2.4.1 A taxonomy of architectures	2-11
2.4.2 Different notions of subtyping	2-11
2.4.3 New compositional mechanisms for systems	2-12

2.4.4 Formalizing architectural connections	2-12
2.4.5 Specifications in the Z language	2-12
2.4.6 Exploiting style in architectural design environments	2-12
2.4.7 Signature matching for software libraries	2-13
2.5 Software Architectures	2-13
2.5.1 Reuse at the software architecture level	2-13
2.5.2 Formalizing descriptions of software architecture	2-14
2.5.3 Architectural styles	2-15
2.5.4 An interchange language for software architecture	2-15
2.5.5 Software architecture primitives	2-16
2.5.6 Research conclusions and directions	2-16
2.5.7 Architecture classification/taxonomy	2-16
2.5.8 Architectural evolution	2-17
2.5.9 Architecture description and analysis	2-17
2.5.10 Reasoning about software architecture	2-18
2.5.11 Architecture description languages	2-19
2.5.12 Development to support ADLs	2-19
2.5.13 Examples and case studies	2-20
2.5.14 Design support: architectural styles	2-21
2.5.15 The software architecture discipline	2-21
2.6 Formal methods	2-22
2.6.1 Structuring Z specifications with views	2-22
2.6.2 Formal specification of concurrent systems	2-22
2.6.3 Specification matching of software components	2-22
2.6.4 Protective interface specifications	2-23
2.6.5 Formal methods: state of the art and future directions	2-23
2.6.6 Hints to specifiers	2-24
2.6.7 Specification matching of software components	2-24
2.6.8 Lightweight formal methods	2-24
2.6.9 Formal methods: state of the art and future directions	2-25
2.6.10 Protection from the underspecified	2-25
2.7 Tractable software analysis	2-25
2.7.1 Formalizing the uni-processor simplex architecture	2-26
2.7.2 Analysis and model checking case studies	2-26
2.7.3 Model checking software systems: A case study	2-27
2.7.4 Theory checking	2-29
2.7.5 Program understanding	2-29
2.7.6 Analysis of software systems	2-30
2.7.7 New analysis mechanisms	2-30
2.7.8 Software engineering	2-31
2.8 Other research	2-32
2.8.1 Safe and efficient persistent heaps	2-32
2.8.2 Language support for mobile agents	2-32
2.9 Bibliography	2-33
3. Integrated Software Architectures	3-1
3.1 Cognitively-Oriented Task Simulation	3-1
3.1.1 Pervasive capabilities for independent task systems	3-1
3.1.2 Speedup learning and large-scale systems	3-8
3.1.3 Soar support	3-10
3.2 High-Performance Planning and Learning	3-12
3.2.1 Plan repair mechanisms	3-14

3.2.2 Learning plan-optimization rules	3-14
3.2.3 Dissemination of Prodigy 4.0 architecture	3-14
3.2.4 Incremental learning of control knowledge for nonlinear problem solving	3-15
3.2.5 Learning control knowledge for plan quality	3-15
3.2.6 Learning domain knowledge by observation and practice	3-16
3.2.7 Planning in a dynamically-changing external world	3-16
3.2.8 Comparison of planning algorithms	3-17
3.2.9 Similarity metrics for case retrieval geometric domains	3-17
3.2.10 Theorem proving by analogy integrated with planning	3-18
3.2.11 Enabling efficient planning technology	3-18
3.2.12 Plan quality	3-18
3.2.13 Learning plan action models	3-19
3.2.14 Learning to reduce search	3-19
3.2.15 Analogical reasoning	3-20
3.2.16 Planning	3-20
3.2.17 Planning with external events	3-21
3.2.18 Real-world domains	3-21
3.2.19 Prodigy-UI	3-23
3.2.20 Learning methods	3-23
3.2.21 Planning algorithms	3-27
3.2.22 Planning, learning, and search algorithms	3-27
3.2.23 Probabilistic planning	3-29
3.2.24 Interleaving planning and execution	3-31
3.2.25 Collaborative, mixed-initiative and adversarial planning	3-31
3.3 Bibliography	3-34
4. Machine Learning in Large Scale Software Environments	4-1
4.1 Reliable indoor navigation	4-1
4.2 Theoretical results for reinforcement learning	4-1
4.3 Probabilistic navigation	4-2
4.4 Learning robot action effects	4-3
4.5 Learning road features for autonomous driving	4-3
4.6 EBNN learning	4-4
4.7 Improving techniques	4-5
4.7.1 Navigation	4-5
4.7.2 Learning visual features	4-5
4.7.3 Autonomously-discovered landmarks	4-6
4.7.4 Obstacle avoidance	4-6
4.7.5 Robot control software	4-6
4.7.6 Testbed	4-7
4.8 Bibliography	4-8
5. Specification, Verification, and Program Development	5-1
5.1 Parallel computing	5-1
5.1.1 First full release of NESL	5-2
5.1.2 The current state of NESL	5-3
5.1.3 Parallel list ranking	5-4
5.1.4 Automatic parallelization of complex scan algorithms	5-5
5.1.5 Theoretical issues in parallel computing	5-5
5.1.6 MPI implementation of CVL and NESL	5-6
5.1.7 NESL implementation of the N-body problem	5-6

5.1.8 NESL implementation of preconditioners	5-6
5.1.9 Parallel computing and program development	5-7
5.1.10 Management of network resources	5-8
5.2 Formal hardware verification and symbolic manipulation	5-9
5.2.1 Extending SMV to use dynamic variable ordering	5-9
5.2.2 Parameterized circuits	5-10
5.2.3 Formal verification of microprocessors	5-11
5.2.4 Image representation and analysis with binary decision diagrams	5-11
5.2.5 Inductive Boolean function manipulation	5-12
5.2.6 A model checker for a VHDL subset	5-12
5.2.7 Symbolic linear-time temporal logic model checking	5-13
5.2.8 Timing issues in circuit verification	5-15
5.2.9 Specification using timing diagrams	5-15
5.2.10 Language inclusion for ω -automata:	5-16
5.2.11 Formal verification of sequential processors	5-17
5.2.12 Inductive Boolean function manipulation	5-18
5.2.13 Design rule checking with BDDs	5-18
5.2.14 Quantitative characteristics of real-time systems	5-19
5.2.15 Counter-examples and witnesses in symbolic model checking	5-21
5.2.16 Extraction of state machines from transistor-level circuits	5-22
5.2.17 Verification of arithmetic circuits	5-23
5.2.18 Verification of arithmetic circuits with binary moment diagrams	5-23
5.2.19 Parallel BDD manipulation	5-24
5.2.20 Reactive system verification	5-24
5.3 Formal verification of sequential processors	5-26
5.3.1 Applications of symbolic trajectory evaluation	5-26
5.3.2 Symbolic representations of discrete functions	5-27
5.3.3 Automatic verification of sequential circuit designs	5-27
5.3.4 Extensions of symbolic trajectory evaluation	5-30
5.3.5 Hierarchical arithmetic circuit verification	5-30
5.4 Parallel programming with NESL	5-30
5.4.1 Implementing nested parallelism	5-31
5.4.2 Applications and algorithms	5-32
5.5 Automatic verification of sequential circuit designs	5-33
5.5.1 Automatic determination of time bounds for sequential circuits	5-33
5.5.2 Reasoning about parameterized circuit designs	5-33
5.5.3 Word-level model checking	5-34
5.6 Interfacing with Java	5-34
5.7 Mapping of NESL onto distributed memory multiprocessors	5-34
5.8 High performance message routing	5-34
5.8.1 All-to-All routing	5-34
5.8.2 Wormhole routing	5-35
5.9 Bibliography	5-36
6. Intelligent Information Integration	6-1
6.1 System design	6-1
6.2 Significance of task assistants	6-2
6.3 Developing task assistants: the Pleiades system	6-2
6.4 Learning reading interests from experience	6-4
6.5 An experience-assisted web agent	6-4
6.6 Warren: A portfolio management system	6-4

6.7 Bazaar: A formal negotiation model	6-5
6.8 A World Wide Knowledge Base	6-5
6.9 Other initiatives	6-6
6.9.1 Active data management	6-6
6.9.2 Matchmaker agents	6-6
6.9.3 Information services	6-6
6.9.4 Self-monitoring	6-6
6.9.5 Reusable agent architecture	6-6
6.9.6 Interest and opportunity matching	6-7
6.10 Bibliography	6-8
7. Spoken-language systems	7-1
7.1 Practical applications of speech recognition research	7-1
7.2 Accomplishments	7-2
7.3 Technology transition	7-3
7.4 Bibliography	7-4
8. Sharing Viewpoints, Objects, and Animations	8-1
8.1 Developing predictive models based on representative tasks	8-1
8.2 Creating interactive programs	8-1
9. Foreign-language Learning	9-1
9.1 Approach	9-1
9.2 Accomplishments	9-2
9.3 Bibliography	9-3
10. Interactive Communication Technology	10-1
10.1 Facilitators for collaboration	10-1
10.2 Accomplishments	10-2
10.3 Bibliography	10-3
11. Multimodal Interaction Technology	11-1
11.1 Multimodal error-repair	11-1
11.2 A multimodal toolkit	11-1
11.3 A multimodal workstation	11-2
11.4 Communicator: A telephone-based dialog system	11-2
11.5 Bibliography	11-4
12. Rapidly Deployable Speech Translation	12-1
12.1 Towards bi-direction language translation	12-1
12.2 Applying DIPLOMAT: Building a translating telephone	12-2
12.3 Implementing the translating telephone	12-2
12.4 Accomplishments in language translation	12-2
12.5 Related accomplishments	12-3
12.6 Technology transition	12-4
12.7 Bibliography	12-5

Introduction

During the period 30 September 1993 to 31 December 1999, Carnegie Mellon pursued a broad set of research initiatives responding to DARPA's Broad Agency Announcement 93-11. As reported here, these projects together constitute a wide-ranging research program that significantly advanced the state of the art in software engineering science and technology.

Our research addressed two fundamental problems of software design and implementation:

- How to ensure that computing systems have the right functionality to meet users' needs, and
- How to make a computing system highly reliable and maintainable over its entire lifetime.

Each of the six original projects attacked one or both of these questions in a different way. In 1997 we began additional projects under the same overall tasking structure. These new efforts included several that applied speech-understanding technology to areas such as multilingual interaction (including translation) and automatic meeting transcription. The following sections provide an overview of the problem space for each of the twelve projects. A comprehensive review of each effort, along with a bibliography that includes selected publication abstracts, follows in subsequent chapters.

1 Team-centered Systems Development

The design of applications is still a craft. We have passed through a burst of innovation in which many of the more obvious tools — word-processors, spreadsheets, etc. — have been built. Rather than waiting for spontaneous innovation to generate another “killer” application, we sought to adopt some of the methodologies of industrial designers and social scientists to develop products based upon thorough understanding of users' needs and capabilities. This project defined the practice of team-centered system development as it applies to a variety of group activities: managing time and information, collaborative writing, and crisis management.

2 Composable Software Systems

Our society has become increasingly depending upon extremely large and complex software systems. As the size of systems has grown over the past 30 to 40 years, software development techniques have advanced enormously, from using primitive machine languages in a bare environment to employing sophisticated programming languages with compilers and module-interconnection tools. We sought to take the next great leap by shifting focus from programming at the *module* level to software development at the *systems* level. We felt it possible to address the engineering and rationalization of many areas of our existing software infrastructure to guide the creation of new megasystems. This project provided (1) a scientific foundation for designing, building, and analyzing large, heterogeneous software systems, (2) architectural languages for describing compositions of software, (3) strategies for dealing with legacy systems, and (4) tools and environments to support our methods and languages.

3 Integrated Software Architectures: Learning, Planning, and Task Simulation

Artificial Intelligence research has been a rich generator of practical applications and products over the years by pushing the envelope of what is possible to program into a computer. Our work on learning and planning had demonstrated enough promise on small examples that sought to create some large systems to work on real domains of interest: mission planning, transportation planning, robotic control, and task simulation.

4 Machine Learning in Large-Scale Software Environments

One key type of knowledge for robotic applications is that of the effects of robot actions. Robot planning approaches that search through the space of possible robot actions rely crucially on such knowledge of action preconditions and effects. However, most experimental work in robotics encounters difficulties in modelling the true preconditions and effects of actions. We have begun exploring the feasibility of learning such action models from experience.

Our research in Machine Learning sought to develop and disseminate a broadly applicable library of machine learning methods and to explore their application to problems in robotics.

5 Specification, Verification, and Program Development for High Performance Computer Architectures

In recent years the parallel processing community has made major steps in simplifying the use of a variety of high-performance parallel and vector computers by supplying languages that port among these machines. Such languages include High Performance Fortran, C*, NESL, and UC. These languages were approaching the point where it is possible to write code once and then run it on a variety of different parallel processors with good runtime efficiency. The languages by themselves, however, do not go far enough to make the machines accessible to nonexperts. To significantly increase the utility of high-performance parallel processors, the community needs to: (1) supply a simple and uniform interface for accessing parallel machines and (2) reduce the time required by users to prototype new parallel algorithms. To address these issues, we are working on an environment and associated language that:

- Allows users to access transparently remote parallel processors as servers,
- Supplies a common debugging environment across diverse machines,
- Allows users to interactively run interactively on parallel machines from their local machine and get immediate results,
- Makes it easy to extend and modify existing libraries,
- Supplies common tools for analyzing running times and optimizing code.

6 Intelligent Information Integration Through Learning and Negotiation

Information is increasingly available from disparate and heterogeneous information sources. The objective of this research is to develop a network of software agents that provide information access and fusion from heterogeneous information sources and to experiment with these agents within a fielded system for supporting a collection of office work tasks. Our thesis is that methods for learning and negotiation will dramatically improve the effectiveness, robustness, scalability, and maintainability of such systems.

7 Spoken-language Systems

We sought to create technology for real-time unlimited vocabulary spoken language processing in the context of practical applications. Our goals included enhancement of the accuracy, robustness, portability, scalability and utility of spoken language systems, through the development of strategies to automatically acquire knowledge at all levels of the process and through unification of structures to represent this knowledge.

8 Sharing Viewpoints, Objects, and Animations

The current state of the art for creating end user applications is to build for a specific delivery mechanism. This is expensive and high risk. Our approach, based on a fundamental understanding of the human perceptual system, is to determine generic, representative tasks (such as object search, object selection, viewpoint manipulation, locomotion) and develop a predictive model. This predictive model will allow us to determine for a given end user application whether or not that application has attributes that merit the expense and difficulty of using a more exotic display mechanism.

9 Foreign-language Learning

We applied automated speech recognition to the acquisition and sustainment of foreign language speaking skills, which are increasingly important in an era of military and civilian multinational endeavors. Language skills are among the most expensive taught in the military, and atrophy quickly when not used. Human language tutors are costly and may be unavailable. Previous language learning software either has not listened to the student, or has been limited in the feedback it provided, typically rejecting spoken utterances without being able to explain what was wrong with them. Our work sought to overcome these limitations.

10 Task-Oriented Communication Technology

We sought to create task-oriented technologies for collaboration among individuals, workgroups, and information services. Such collaborations will be mediated by computers to enhance access to relevant information or to overcome barriers of time, space, or culture. The key to such collaborations is communication in the context of information relevant to the task at hand. This means that collaboration must occur in the context of information tailored to the task at hand. Collaboration must also be able to cross language barriers, and all interactive software must support collaboration with relevant parties. Collaboration must not be confined to specialized software ghettos and must make effective use of all human communication modalities that can be applied to the task at hand (e.g., speech, gesture, handwriting, face-, eye-, pose-, body-motion). Finally, collaboration technologies must adapt to the tasks and the people involved, rather than the other way around.

11 Multimodal Interaction Technology

Our research in this area concerned cross-modal repair of errors in automatically recognized speech. Previous research in speech-recognition systems had not sufficiently addressed the problem of recognition errors. Despite a few successful commercializations, e.g. dictation systems for computer users with repetitive-strain injuries, informal user surveys suggested that

speech-recognition repair methods of the time were insufficient. These methods include repeating speech-input (“respeaking”), choosing from interpretation alternatives, and using the keyboard. We sought to provide much more efficient repair mechanisms for automatic speech recognition.

12 Rapidly Deployable Speech Translation

DIPLOMAT is a pilot project in rapid-deployment, unrestricted speech-to-speech translation. We sought to make this system retargetable to both new languages and new domains orders of magnitude more quickly than commercial technology.

1. Team-centered System Development

1.1 User-centered design of group planning tools

Calendar and time management systems are key artifacts to study in the area of group work and coordination. They encapsulate the state-of-practice for any community that uses them.

There are many well-proven, industrial-strength computational tools for planning: PERT charts, Gant Charts, PERT-COST systems, Heuristic Planning systems, etc. There have also been many attempts to solve the group scheduling and coordination problem through the use of computers. They often fail for completely understandable reasons that have nothing to do with their excellently rational designs. Most people jealously protect the scheduling of their time and the control of their activities. Unless they are strongly motivated to serve a central authority, they exercise a great deal of personal discretion about what tasks they undertake and when.

The weakest link in the group coordination activity is the interaction of an individual with the group. It is computationally easy to plan, schedule, and track a set of tasks associated with a single, well-defined project. However, it is very hard to get the attention and commitment of all the individual actors in that project so that their contributions are timely and relevant. The typical individual in a modern organization might simultaneously be involved with several projects each with a different cast of characters and having different goals. They must make daily decisions of what to attend to and what level of effort to expend.

The spirit of user-centered design suggests that if one is designing a group calendar-coordination system that she start not with the central scheduler algorithm, but with the individual user and her perception of the "system". An individual's entry to a group coordination system should be a personal calendar system; for most people that is the most natural and familiar planning tool.

This indirect approach has another crucial benefit: there is a motivation for individual adoption of the tool long before any group commitment is required. If a manager looks around some month and discovers that half her organization is using, e.g. the Franklin Planner, she has already solved the "adoption" problem and can proceed to build on the common vocabulary and practices of that system.

1.1.1 Preliminary activities

We began this research with a knowledge-gathering and planning effort: a literature search, informal interviews of putative users, speculation, and brainstorming. Some ideas from the literature that have influenced our design include:

- The artifacts and details of time management are less important to an individual or group than clarity of goals and recognition of constraints. No time management system can succeed unless its manager has a deep understanding of purpose.
- Many people, organizations, and societies, are not calendar or task driven but give primacy to human relationships and transactions. Salesmen, lawmakers chief-executives, and other powerfully productive people treat calendars as the small tail of a big dog.
- Some of the characteristics of recognized star performers are: taking initiative, and

regulating one's own work commitments, time, performance level, and career growth. In other words, they control their own activities and calendars.

- Performance can be divided into effort, results, and impact. Prospective or retrospective calendars measure only the first of these.
- Paper calendars are preferred to many computer systems because they allow a user to see lots of context when they are scheduling an event and allow at-a-glance browsing.
- To-Do lists, both long-term master lists, and short-term working lists are written in various places on or around calendars.
- Goals and tasks have different kinds of temporal constraints, some are tightly scheduled, some loosely. Some intentions are temporally constrained by other intentions. There is a tendency for the timely to drive out the important.

We have examined a few computer calendar systems:

- The most popular tools, e.g. Now-Up-To-Date, are good simulations of the paper systems with some computer benefits thrown in.
- Rigorously and rigidly organized systems, e.g. Ascend, are less popular even with people who consciously use a highly structured planning system, e.g. Franklin Planner. A common statement by users of the more normative systems was, "It's great, but I only use about half of it."

1.1.2 Informal interviews

We interviewed several people at length regarding their time-management behavior. From these interviews, we gathered the following insights:

- Even the best computer calendar systems don't permit as much flexibility as some users might want. Some of the problems users mentioned were that the system does not permit dragging between weeks, and that a manager might want to be able to track the frequency of his contacts with key individuals or work groups.
- A "why" exercise—asking repeatedly of a week's activities, "Why, did/would you do this?"—revealed about 17 different one-word reasons for a typical professional. A surprise, obvious in retrospect, was that many things are done to maintain or enhance an important personal relationship.
- People may schedule time more on the basis of how tasks break up into chunks than by explicit schedules or deadlines. For example, if I see that I only have 30 minutes before my next meeting, I will select some task which I know I can achieve sufficient closure in within 30 minutes or less.
- We need support for the link between small tasks and long-term tasks. For example, writing an NSF proposal is a fairly important task, but it involves so much work over such a long period of time, that simply putting a "Write NSF proposal" item in a to-do list or on a calendar does offer much help in scheduling the time to do it. We may want our system to have a priori knowledge of the decomposition of some high-level tasks (including information about temporal dependence) so that we can provide the users with a set of low-level tasks which can then be scheduled.

1.1.3 System goals

It is unclear exactly what the measurable goals of a time management system are. At the ridiculously narrow end the goal might be that a person never miss a meeting or decide in the shortest time which task on her to-do list to undertake. At the ridiculously vague end the goal might be whether the person is being effective in their job or fulfilled in her life. We know the phenomena we're looking for: The individual or group doesn't necessarily produce more, but they are quicker to make or refuse commitments, more likely to complete committed tasks in a timely manner, more likely to undertake preparations for upcoming events, and better able to explain why a particular task is being performed at a particular time.

A possible test group for this system is college students. They have many explicitly scheduled obligations and bespoke goals. The artificiality of their situation actually makes analyzing their behavior easier.

From a detailed study of the weekly recorded activities of a university professor we found that many activities have an unclear pay-off but are nonetheless valuable. Often the rationale is something like, "Maybe X will buy my idea if I meet him at the Y meeting." One of the main complications of scheduling is estimating the positive and negative outcomes of various activities. "If I spend an hour in the library I might find the answer to question X." "If I skip this lecture I might miss the answers to the quiz." In fact, tasks with known outcomes are easy to handle but often uninteresting. There may be value in a user prospectively and retrospectively rating the importance of tasks. Another frequent explanation for tasks was that they were "owed" to someone else. A distinct transactional aspect to the person's activities emerged. "I go to this meeting because X needs me there and X has helped me with the Y problem." Thus the "why's" of various tasks are often not based on cause/effect relationships between goals and sub-goals but on a kind of vague commerce in human obligations which may or may not lead to tangible benefits or penalties.

We created many pictures of tasks and schedules. The idea was to explore a wide range of metaphors for describing the set of activities and goals facing an individual. A general observation we made was that a person will react more immediately to a size or shape of a task than a written description or numerical value. When an unimportant task happened to take up a lot of space in a picture, the person would react strongly.

1.2 The prototype

We created a initial prototype calendar system from a set of preliminary design ideas. Its major purpose was to elicit some reactions from potential users. The main ideas included:

- **Making Goals Salient** To-do lists typically are a mixture of potential activities, like "walk the dog" or "move the division", and goals like "find baby-sitter" or "take the hill". A goal is different from an activity because the amount of time or resource to achieve it is unknown.

The prototype provided a panel for explicitly displaying goals and showing the relation among them. Achieved goals can be checked off.

- **Mixed initiative planning** The mock-up looks like a paper calendar system except that items are pasted, like post-it notes, onto the display rather than written. There are very few constraints on where one pastes a task or appointment, or on what it

looks like. There are (breakable) constraints on the beginning and ending times of tasks, and computing power is used mainly to analyze effort levels, point out conflicts, and suggest schedules. In other words, the user always has the power to override any computer-made plans and can easily adjust any constraints or parameters.

- **Meaningful graphics** The graphics should "work" in Tufte's sense rather than be merely decorative. The size, shape, color and placement of graphic elements have meanings for the scheduling process. For example, height represents task duration. This has two purposes: to minimize the typing a person must do to record and adjust their plans and to make "at-a-glance" analysis of a plan easier.
- **Zooming and depth** Relating the "big picture" to the daily detail is an crucial aspect of planning. The mock-up allows one to view the schedule from a variety of virtual distances as an alternative to the day-week-month viewing alternatives used by most systems. The placement of data in three-dimensional space might be tried.
- **Deeper structure** Any real calendar or schedule is the outcome of a deeper process involving goals and constraints. An attempt should be made to make these less overt decisions explicit in the tool and relate them to actual activities. Many calendar systems make an attempt at this feature but none has succeeded. Some have no "point-of-view" and others are too rigid.

1.3 Research questions

This project is aimed at creating an artifact: a time and goal management calendar system. However, it is also aimed at getting some general answers to some fundamental questions of human computer interaction:

- How to make input capture easy enough
- What are the important features of tasks
- What is user-centered design.

1.4 Prototype development and testing

We created a subsequent prototype calendar system (on a Mac) that embodied a number of new features:

- The system categorized items according to their corresponding goal or job.
- Their shape/color of an item was determined by its category.
- The user could move planned items (horizontally) within a time period to reflect the likelihood that they would actually be performed. Past items not performed stayed on the calendar for future analysis.
- For any period, the user could compute a tally of activities, providing a pie chart suggesting the user's effort allocation during that period.

We simulated several graphical displays comparing various aspects of tasks, e.g. urgency vs. importance. Several users, most previously familiar with Now-Up-To-Date, informally tested the prototype. These users reported that its features helped them understand and better utilize the calendar-planning process

1.5 Bibliography

[Kamas and Reder 94]

Kamas, E., and L.M. Reder.

The role of familiarity in cognitive processing.

Sources of coherence in text comprehension: A festschrift in honor of Jerome L. Myers.

In O'Brien, E. and R. Lorch,

L. Erlbaum, 1994.

[Kosbie and Myers 94]

Kosbie, D.S., and B.A. Myers.

Extending programming by demonstration with hierarchical event histories.

Technical Report CMU-CS-94-156, Computer Science Department, Carnegie Mellon University,

May, 1994.

Also appears as Human Computer Interaction Institute Technical Report CMU-HCII-94-102.

[Landay and Myers 94]

Landay, J.A., and B.A. Myers.

Interactive sketching for the early stages of user interface design.

Technical Report CMU-CS-94-176, Computer Science Department, Carnegie Mellon University,

July, 1994.

Also appears as Human Computer Interaction Institute Technical Report CMU-HCII-94-104.

[Lebiere et al. 94]

Lebiere, C., J.R. Anderson, and L.M. Reder.

Error modeling in the ACT-R production system.

In *Proceedings of the Cognitive Science Society*. CSS, 1994.

To appear.

[Miner and Reder 94]

Miner, A., and L.M. Reder.

A new look at the feeling of knowing: Its metacognitive role in regulating question answering.

Metacognition: Knowing about knowing.

In Metcalfe, J., and A. Shimamura,

MIT Press, 1994.

[Myers 94a]

Myers, B.A.

User interface software tools.

ACM Transactions on Computer-Human Interaction, 1994.

To appear.

[Myers 94b]

Myers, B.A.

Challenges of HCI design and implementation.

*ACM Interactions*1(1):73-83, January, 1994.

[Myers 94c]

Myers, B.A.

The Garnet user interface development environment: Demonstration abstract.
In *CHI'94 Conference Companion*, pages 25-26. CHI, April, 1994.

[Myers and Olsen 94]

Myers, B.A., and D.R. Olsen, Jr.

User interface tools: Tutorial description.

In *CHI'94 Conference Companion*, pages 421-422. CHI, April, 1994.

[Myers et al. 94a]

Myers, B.A., D. Giuse, A. Mickish, B. Vander Zanden, D. Kosbie, R. McDaniel, J. Landay, M. Goldberg, and R. Pathasarathy.

The Garnet user interface development environment: Video abstract.

In *CHI'94 Conference Companion*, pages 455-456. CHI, April, 1994.

[Myers et al. 94b]

Myers, B.A., D.A. Giuse, A. Mickish, and D.S. Kosbie.

Making structured graphics and constraints practical for large-scale applications.

Technical Report CMU-CS-94-109, Computer Science Department, Carnegie Mellon University,

May, 1994.

Also appears as Human Computer Interaction Institute Technical Report CMU-HCII-94-100.

[Reder and Gordon 94]

Reder, L.M., and J.S. Gordon.

Subliminal perception: Nothing special cognitively speaking.

Cognitive and Neuropsychological Approaches to the Study of Consciousness.

In Cohen, J., and J. Schooler,

L. Erlbaum, 1994.

In press.

[Reder and Klatzky 94]

Reder, L.M., and R. Klatzky.

Transfer: Training for Performance.

Learning, Remembering, Believing: Enhancing team and individual performance.

In Druckman, D., and R.A. Bjork,

National Academy Press, 1994.

In press.

[Schooler et al. 94]

Schooler, J.S., R. Ryan, and L.M. Reder.

The cost and benefits of verbally rehearsing memory for faces.

Basic and Applied Memory Research, Vol. II.

In Herrmann, D.J., M.K. Johnson, C. Hertzog, C. McEvoy, and P. Hertel,

L. Erlbaum, 1994.

In press.

[Vander Zanden et al. 94]

Vander Zanden, B., B.A. Myers, D. Giuse, and P. Szekely.

Integrating pointer variables into one-way constraint models.

ACM Transactions on Computer-Human Interaction 1(2), June, 1994.

2. Composable Software Systems

Software techniques have developed enormously over the past 30-40 years, from primitive machine languages to sophisticated programming languages and tools for system configuration. We have worked on the next great leap, shifting focus from module-level programming to system-level programming. In the former activity we had built programs in terms of procedures and abstract data types using simple module interconnection as a way to compose modules. In the latter, we build systems in terms of more sophisticated components, usually entire systems themselves, and connect them with more sophisticated abstractions for composition.

What makes the construction of composable systems different from programming?

- We are liberating ourselves from thinking of the task as merely programming. We are not just building a program, we are building a system.
- Our units of manipulation are components and connectors, rather than simply lines of source code. Our innovative claim is that connectors are first-class entities — just as components are — in a system.
- We want to provide ways to talk precisely about common patterns of structures. This allows us to exploit special properties of particular composition idioms for analysis, design guidance, and efficient implementation.

The research program at Carnegie Mellon in Composable Software Systems has had two primary goals:

1. To provide a scientific and engineering basis for designing, building, and analyzing composable systems;
2. To provide languages, tools, environments, and techniques to support (1).

Three themes cut across our individual research projects and interests: software architectures (Garlan, Shaw), formal methods (Garlan, Jackson, Wing), and tractable software analysis (Jackson). The report of our progress is organized around these three themes.

This report covers work performed from the beginning of the contract period through June 1997. Work on these and related issues then continued under a separately-funded project, "A Technology Investigation Supporting Software Architecture and Analysis for Evolution."

2.1 Scientific foundation

To establish a scientific foundation—in the form of semantics, descriptions, and analyses—for building composable systems, we must understand the limitations of current models, theories, and methods of software construction while developing new, more appropriate ones. Using examples from practical systems in our research, our major goal is to develop methods and techniques for classifying, specifying, analyzing, and designing software systems.

2.1.1 In search of standard software architecture problems

Many disciplines use standard problems to focus discussion. If the problem is well-understood, discussion can proceed to solution techniques rather than hanging up on problem details. In an effort to foster progress in the software architecture community, we are developing a collection of good, shared problems. We circulated the first version at the International Workshop on

Software Specification and Design. We converted our earlier, paper document to a web version in [Shaw et al. 95a]. Since our goal is developing community consensus rather than final publication, we expected this to pass through many versions circulated within the community

Model problems and research directions in software architecture

The software architecture community would benefit from sharing a set of standard example problems. This set of problems, such as the collection we've been assembling, would improve our ability to work out ideas, exhibit techniques, and compare results. We intend to stimulate a discussion about suitable problems: What characteristics should they have, what specific problems would serve us well? We converted our earlier, paper document to a web version in [Shaw et al. 95a]. We expect smoother updates and wider distribution this way.

[Garlan 95a] outlines the challenging research problems in Software Architecture.

2.1.2 Software development education

Part of the transition plan for this grant involved education. We made progress disseminating our material on how to teach software architecture principles by producing a report describing software development exercises for a software architecture course. [Garlan and Shaw 94a] presents a set of exercises developed for a course taught at Carnegie Mellon.

We have developed and offered a course (and associated course materials) in software development. The software development exercises we developed for this course give students hands-on experience with distinctly different architectures. We presented these assignments at the software engineering education workshop at ICSE-16 in May [Garlan and Shaw 94b].

2.1.3 Developing a formal basis for architectural style

Developers describe the software architecture of most systems informally and diagrammatically. For these descriptions to be meaningful at all, one must understand figures by interpreting the boxes and lines in specific, conventionalized ways. In [Abowd et al. 93] we treat these conventionalized interpretations as architectural "styles" and provide a formal framework for their uniform definition.

2.1.4 Formal specification of AEC product models

[Chadha et al. 94] illustrates the use of equational specifications in developing product data models. This approach enables a precise and abstract description of products, where both syntactic and semantic checks are used for validation. Because they are formal objects, these specifications can be validated with respect to formal requirements and combined using ordinary mathematics. In addition, the availability of mature tools from the software engineering community further supports this approach to specifying and validating product models. To the best of our knowledge, this is the first application of formal methods in civil engineering.

2.1.5 Decomposing and recomposing transactional concepts

In the Venari Project we teased apart the usual atomicity, serializability, and persistence properties rolled into transactions, and added the ability for transactions to be multi-threaded. In particular, we provide support for the following features, each as a separable component: persistence, undoability, reader/writer locks, threads, and skeins. We want the individual pieces to compose in a seamless way to give us transactions. Persistence ensures permanence of effects of top-level transactions. Undoability allows us to handle aborted transactions. Reader/writer locks provide isolation of changes to the store, and hence ensure transaction serializability of concurrent transactions. Skeins let us group a collection of threads together, giving us the ability to make multi-threaded transactions. [Wing 94] focuses on the details of the synchronization primitives we provide for our model of computation.

2.1.6 Semantics for parallel architectures

We explored the use of formal semantics to reason about the behavior of software designed to be executed on a parallel architecture. One aim of this research was to develop mathematical tools to formulate common parallel program design styles and to make it easier to design and analyze parallel programs.

2.1.7 Coping with heterogeneity in software architecture

For software, as for buildings, no single architectural style can solve all problems: Heterogeneity is inevitable. Just as inevitably, diverse components and systems will have to work together. Distinct architectural styles often require different component packaging and interactions; these complicate the interoperation problem. We need to improve our ability to recognize mismatches among heterogeneous parts, to organize our current ad hoc techniques for coping with these mismatches, and to develop design guidance for selecting the appropriate mismatch resolution technique for each specific problem. [Shaw 95a] lays out a preliminary structure for discussing the problem and suggests useful directions.

2.1.8 A software design paradigm based on process control

A standard demonstration problem in object-oriented programming is the design of an automobile cruise control. This design exercise demonstrates object-oriented techniques well, but it does not ask whether the object-oriented paradigm is the best one for the task. In [Shaw 94a] we examine the alternative view that cruise control is essentially a control problem. We present a new software organization paradigm motivated by process control loops. The control view leads us to an architecture that is dominated by analysis of a classical feedback loop rather than by the identification of discrete stateful components to treat as objects. The change in architectural model calls attention to important questions about the cruise control task that aren't addressed in an object-oriented design.

2.1.9 A software systems architectures course

Previously, we summarized experience with a course taught at Carnegie Mellon on architectures for software systems. This course serves as an important transition vehicle for some of the Composable Systems research. A follow-on report [Garlan et al. 94a] collects and makes public the course materials, including lecture slides and assignments.

2.1.10 Architectural mismatch

Many would argue that future breakthroughs in software productivity will depend on our ability to combine existing pieces of software to produce new applications. An important step towards this goal is the development of new techniques to detect and cope with mismatches in the assembled parts. Some problems of composition are due to low-level issues of interoperability, such as mismatches in programming languages or database schemas. However, in [Garlan et al. 95] we highlight a different, and in many ways more pervasive, class of problem: *architectural mismatch*. Specifically, we use our experience in building a family of software design environments from existing parts to illustrate various mismatch types centering around the assumptions a reusable part makes about the structure of the application in which it is to appear. Based on this experience we show how an architectural view of the mismatch problem exposes some thorny, fundamental problems for software composition and suggests possible research avenues to solve them.

2.1.11 Using architectural style

A central aspect of architectural design is the use of recurring organizational patterns and idioms, or *architectural styles*. Unfortunately, the use of architectural styles is almost completely ad hoc. What is needed is a more rigorous basis for understanding architectural style and ways to exploit it. In [Garlan 95b] we briefly outline and compare three approaches to providing such a basis.

2.1.12 Subtyping in object-oriented languages

The use of hierarchy is an important component of object-oriented design. Hierarchy allows the use of type families, in which higher level supertypes capture the behavior that all of their subtypes have in common. For this methodology to be effective, it is necessary to have a clear understanding of how subtypes and supertypes are related. In [Liskov and Wing 94a], we take the position that the relationship should ensure that any property proved about supertype objects also holds for its subtype objects. We present two ways of defining the subtype relation, each of which meets this criterion, and each of which is easy for programmers to use. The subtype relation is based on the specifications of the sub- and supertypes; we present a way of specifying types that makes it convenient to define the subtype relation. We also discuss the ramifications of this notion of subtyping on the design of type families.

2.1.13 Composing transactional concepts

In [Haines et al. 94] we describe the design of a transaction facility for a language that supports higher-order functions. We factor transactions into four separable features: persistence, undoability, locking, and threads. Then, relying on function composition, we show how we can put them together again. Our modular approach towards building transactions enables us to construct a model of concurrent, nested, multi-threaded transactions, as well as other non-traditional models where not all features of traditional transactions are present. Key to our approach is the use of higher-order functions to make transactions first-class. Not only do we get clean composability of transactional features, but also we avoid the need to introduce special control and block-structured constructs as done in more traditional transactional systems. We implemented our design in Standard ML of New Jersey.

2.1.14 Applying formal methods to distributed systems

The notion of belief has been useful in reasoning about authentication protocols. In [Mummert et al. 94] we show how the notion of belief can be applied to reasoning about cache coherence in a distributed file system. To the best of our knowledge, this is the first formal analysis of this problem. We used an extended subset of a logic of authentication [Burrows, Abadi, Needham] to help us analyze three, cache-coherence protocols: a validate-on-use protocol, an invalidation-based protocol, and a new large granularity protocol for use in weakly-connected environments. We present two runs from the large granularity protocol. Using our variant of the logic of authentication, we were able to find flaws in the design of the large granularity protocol. We found the notion of belief not only intuitively appealing for reasoning about our protocols, but also practical, given the optimistic nature of our system model.

2.1.15 Automated analysis of design

In [Jackson 94a] we show how properties of a design that involve undecidable claims about unbounded objects (typical in software designs) can sometimes be determined automatically by model-checking techniques. The method involves the application of abstractions to different datatypes and a novel use of symbolic abstraction to compare the contents of sets and relations.

[Jackson 94b] introduces a promising new reduction technique that can reduce by a large factor the search space for checking a software design. Using symmetry, the paper shows how finding flaws in a small specification of a phone switch can reduce the state space search by a factor of up to 100. As the size of the state space increases, the reduction factor increases, too: In a more recent example we obtained a reduction of over 5,500. This reduces an hour's analysis to less than a second and a year's analysis to less than two hours, raising the prospect of automatically checking designs that have not previously been regarded as amenable to automatic checking.

2.1.16 Specifying weak sets

In [Wing 95a, Wing and Steere 95] we present formal specifications of a new abstraction, *weak sets*, that can be used to alleviate high latencies when retrieving data from a wide-area information system such as the World Wide Web. In the presence of failures, concurrency, and distribution, clients performing queries may observe behavior that is inconsistent with the stringent semantic requirements of mathematical sets. For example an element retrieved and returned to the client may be subsequently deleted before the query terminates. We chose to specify formally the behavior of weak sets because we wanted to understand the varying degrees of inconsistency clients might be willing to tolerate and to understand the tradeoff between providing strong consistency guarantees and implementing weak sets efficiently. Our specification assertion language uses a novel construct that lets us model *reachability* explicitly; with this language, we can distinguish between the existence of an object and its accessibility. These specifications were instrumental in understanding the design space, and we are currently implementing the most permissive of the specifications in several types of UNIX system.

2.1.17 Abstract models of memory management

Most specifications of garbage collectors concentrate on the low-level algorithmic details of *how* to find and preserve accessible objects. Often, they focus on bit-level manipulations such as “scanning stack frames,” “marking objects,” “tagging data,” etc. While these details are important in some contexts, they often obscure the more fundamental aspects of memory management: *What* objects are garbage and *why*?

We developed a series of calculi that are just low-level enough that we can express allocation and garbage collection, yet are sufficiently abstract that we can formally prove the correctness of various memory management strategies [Morrisett et al. 95]. By making the heap of a program syntactically apparent, we can specify memory actions as rewriting rules that allocate values on the heap and automatically dereference pointers to such objects when needed. This formulation permits the specification of garbage collection as a relation that removes portions of the heap without affecting the outcome of the evaluation.

Our high-level approach allows us to specify, in a compact manner, a wide variety of memory management techniques, including standard trace-based garbage collection, (i.e., the family of copying and mark/sweep collection algorithms), generational collection, and type-based, tag-free collection. Furthermore, since the definition of garbage is based on the *semantics* of the underlying language instead of the conservative approximation of inaccessibility, we are able to specify and prove the idea that type inference can be used to collect some objects that are accessible but never used.

2.1.18 Polymorphic closure conversion

We studied the typing properties of *closure conversion* for the simply-typed and polymorphic λ -calculi [Minamide et al. 96]. Unlike most accounts of closure conversion that only treat the untyped λ -calculus, our account translates well-typed source programs to well-typed target programs. This allows later compiler phases to exploit types and facilitates proving correctness. Our account of closure conversion for the simply-typed language takes advantage of Pierce and Turner’s simple model of objects by mapping closures to *existentials*. Closure conversion for the polymorphic language requires additional type machinery, namely *translucency* in the style of Harper and Lillibridge’s module calculus, to express the type of a closure.

2.1.19 An architecture for a family of instrumentation systems

An important challenge for formal methods is to demonstrate their ability to scale to cost-effective specification of large-scale systems. In [Garlan and Delisle 95a] we describe our experience in applying formal methods to developing of a family of industrial products. The approach addresses problems of scalability by focusing on an architectural level of abstraction—concentrating on overall system organization and clean compositional techniques for a family of designs, rather than on the complete specification of a particular system.

2.1.20 View structuring

Specifications have traditionally been decomposed hierarchically into components. For requirements engineering in particular, it is often more useful to decompose into parallel "views." Instead of organizing the specification according to functional components, different aspects of the system are separated into different views. For example, a word processor might have a line-based view to describe scrolling, a word-based view to describe spelling checks and word search/replace, a grid view to describe cursor motion, and so on.

View structuring gives a better separation of concerns than traditional structuring styles and offers many advantages. It allows simpler analysis, since many properties can be evaluated within a single view. It is especially helpful in requirements elicitation, because views can be added incrementally, and different views can record the functionality of the system as required by different users. It may also lead to more reuse, since view structuring disentangles different aspects of a system's function, that otherwise tend to complicate the structure of components and prevent reuse.

[Jackson 95a] shows how the **Z** specification language can support view structuring in a simple and direct fashion, discussing the advantages and disadvantages of **Z** for this purpose. In this reporting period we made extensive improvements to the paper based on referees' comments. We expect the paper to be published in the next few months.

2.1.21 Architectural issues in software reuse

One of the major impediments to effective software reuse is the implicit nature of the assumptions about how components interact with each other. When these don't match, system integration is unduly complex. In [Shaw 95b, Shaw 95a] we examine the problem and start to analyze what people actually do about it.

2.1.22 Teaching mathematics to software engineers

Based on experience in teaching formal methods to practicing and aspiring software engineers [Wing 95b], updated in [Wing 96], presents some of the common stumbling blocks faced when writing formal specifications. The most conspicuous problem is learning to abstract. We address all these problems indirectly by giving a list of hints to specifiers. Thus this paper should be of interest not only to teachers of formal methods, but also to their students.

2.2 Architectural languages

The software composition problem has all the elements that make language design an appropriate vehicle for a solution: elements, composition rules, abstractions, and closure. The components and connectors of various kinds are the elements. The rules that determine how systems can be built out of components and connectors are the composition rules. Idiomatic usage in existing systems shows the need for abstractions, and closure rules are needed to determine whether an arbitrary composition can be used. We are developing both general-purpose architectural languages and specialized languages for different compositional styles, application-oriented families of systems, and specific compositional patterns.

As the size and complexity of software systems increases, the design and specification of overall

system structure—software architecture—emerges as a central concern. Architectural issues include the gross organization of the system, protocols for communication and data access, assignment of functionality to design elements, and selection among design alternatives. [Shaw et al. 96] explains the inner workings of connector implementation. [Shaw et al. 95b] describes abstractions for software architecture and tools to support such abstractions.

Currently system designers have at their disposal two primary ways of defining software architecture: They can use the modularization facilities of existing programming languages and module interconnection languages, or they can describe their designs using informal diagrams and idiomatic phrases, such as “client-server organization.”

In [Shaw and Garlan 94] we explain why neither alternative is adequate. We consider the nature of architectural description as it is performed informally by systems designers. Then we show that regularities in these descriptions can form the basis for architectural description languages. Next, we identify specific properties that such languages should have. Last, we illustrate how current notations fail to satisfy those properties.

2.2.1 A theory and description language for component interactions

A formal basis for describing and analyzing architectural designs represents an important step towards an engineering discipline for composable systems. In [Allen and Garlan 94a, Allen and Garlan 94b, Garlan 94] we present a theory for one aspect of architectural description: the interactions between components. The key idea is to define architectural connectors as first-class semantic entities. These entities are specified as a collection of protocols that characterize each of the interaction participant roles and how these roles interact. In these papers we illustrate how this scheme can be used to define a variety of common architectural connectors. We provide a formal semantics and show how this leads to a sound deductive system in which architectural compatibility can be checked in a way analogous to type checking in programming languages.

2.2.2 A constraint-based, object-oriented, programming language

[Horn 93] presents a new model of programming, called constrained objects, in which algebraic constraints are used as a foundation for object-oriented programming. In this model objects are encapsulated constraint systems that communicate with each other via message-passing. Each object maintains a consistent state under perturbation from messages sent by other objects using a constraint satisfaction process. One object may inherit from another: The set of constraints from the first object is conjoined with the set of constraints from the second, and the subsequent satisfaction of the combined constraints ensures the semantic consistency of the derived object.

2.2.3 Patterns for software architectures

The patterns community is exploring ways to explain software design strategies. We’re investing energy in keeping contact between the architecture community and the pattern community—their objectives are similar in many respects. [Shaw 95c] sketches pattern-like descriptions of architectural styles.

[Shaw 94b] appeared as a book chapter in Spring 1995. It appears that this book will be popular in the patterns community. This work is another part of the ongoing effort to interact with the patterns community.

2.2.4 Formalizing architectural style

The software architecture of most systems is usually described informally and diagrammatically by means of boxes and lines. To make these descriptions meaningful, the diagrams are understood by interpreting the boxes and lines in specific, conventionalized ways. The informal, imprecise nature of these interpretations has a number of limitations. In [Abowd et al. 95] we consider these conventionalized interpretations as architectural styles and provide a formal framework for their uniform definition. In addition to providing a template for precisely defining new architectural styles, this framework allows for analysis within and between different architectural styles.

2.2.5 Re-design of UniCon

We completed the design and implementation of the initial version of the UniCon language (Shaw&.94.Abstractions). At the beginning of this, we began a second version. The purposes of the system and reimplementations are (1) to take advantage of what we learned in the first version and make it straightforward to add new components and connectors, (2) to substantially change the graphical notation, (3) to integrate graphical interface and external tools properly with the system, and (4) to resolve some difficulties building hierarchical systems and change the system builder "make" to Odin.

2.3 Legacy systems

Many legacy systems represent a huge investment, and maintainers cannot afford to completely rewrite such pre-existing software to enable using new tools and methods. Bringing legacy systems into our architectural framework will benefit people who maintain them: It will enable them to analyze existing systems and understand them as well as new ones. This knowledge will provide support for connecting several existing systems into larger systems. Secondly, we anticipate that accommodating legacy systems may lead to results and insights that will be useful in new systems.

We are working on two major tasks related to legacy systems. The first task involves using automatic analyses to understand a system's structure and recover its architectural design, which is usually not expressed directly. With this information we generate architectural views for specific purposes, such as a particular analysis or connection. These views reveal relationships between components and indicate how parts of the system's functionality are distributed throughout its components. They also indicate how to identify the components involved in discharging particular functional requirements.

2.3.1 Aspect: detecting bugs with abstract dependences

"Aspect" is a tool for detecting bugs in code, using partial formal specifications against which the code can be analyzed automatically. In moving towards applying these ideas to larger systems issues, we have focused on the use of abstraction in Aspect and shown how a complex editor buffer can be analyzed in terms of simple specifications not much longer than type declarations.

[Jackson 94c] rigorously explains the basis of Aspect and completely recasts the semantic model

in the mold of an Algol-like language (where previous work assumes the semantic model of LISP).

2.3.2 Viewpoint composition in model-based specifications

Other work focuses on requirements specification. We are investigating a new approach to *structuring* a specification. Rather than dividing a system into modules and specifying them separately, thus giving complete descriptions of parts of the system, we construct the specification as a collection of views, each being a partial description of the entire system. The idea of views is not new, but ours is the first systematic exposition of the idea in the context of a conventional, model-based specification language such as Z.

2.3.3 Reverse engineering

In collaboration with AT&T we have developed a technique to generate automatically a report describing the differences between two versions of a procedure. We have built a prototype tool and tested the idea on examples from the AT&T Autoplex system. Related work on semantic differencing has focused on highlighting portions of code that are affected by any change, however slight. As a result even as small a change as fixing an off-by-one error can appear to have wide repercussions. Our work, on the other hand, uses a novel representation of code, based on Aspect, to report only the differences that have *architectural* significance.

The larger part of our research into reverse engineering involves the construction of a tool to answer queries about C code. We sought to build a reverse-engineering tool that provides an editor interface, and answers semantic queries that no other then-current tool could handle, such as “where was this variable—the one identified by the cursor—set?”, or “where is the procedure that used this global variable to affect this field of this structure?” We wished to demonstrate a small set of powerful but low-level queries and then showed that more abstract architectural analyses can be built as editor macros on top of these.

2.3.4 Advances in reverse engineering

We performed a series of experiments—with early versions of our reverse engineering tool and on paper—and devised a new way to extract partial views of a program, a method that offers several advantages over existing techniques (such as slicing).

We invented and implemented a new program representation and algorithms to support this extraction process [Jackson and Rollins 94a]. We then worked to migrate this first, inefficient prototype into a more robust tool.

We also completed a paper [Jackson and Ladd 94] describing a tool that generates a report of the semantic consequences of changes to code.

We have completed a prototype implementation of Chopshop, a reverse engineering tool for C programs. Chopshop handles the entire ANSI C language. It generates program slices in textual and pictorial form.

Existing program slicers have focused on statement-level analysis. For large systems, this ap-

proach fails: The slices are too big, and crucial information is not presented. In particular, a slice does not indicate why a procedure is included. Chopshop performs a fine-grained, statement-level analysis (thus retaining semantic accuracy), but can present its results at the module-level. The user can select a data structure, or a small part of a data structure, and ask what influences its value or what it influences. Chopshop, in response, can highlight code and display pictures showing the procedures that are relevant, along with information about why they are relevant.

[Jackson and Rollins 94b] presents the model on which Chopshop is based. It makes two primary contributions. First, it shows how interprocedural slices can be calculated in a modular fashion, using simpler algorithms than existing techniques and obtaining more accurate results. Second, it shows how the role of procedure calls in a slice can be summarized and presented.

[Jackson and Rollins 94c] presents Chopshop's filtering mechanisms. It shows, using the code of a UNIX utility as an example, how global information about the role of procedures in a slice can be presented in a picture, while hiding local information that can be determined easily by examining code in the vicinity of the query.

We developed a "call graph slicer" based on the Chopshop tool that extracts slices of large C programs. Unlike a conventional slicer that takes a query about the appearance of a variable in some statement, and generates as output a subprogram, giving all statements that might affect the chosen variable, our slicer operates on the call graph. The user selects an argument to a procedure; the tool displays the relevant portion of the call graph with arrows between procedures indicating relevant dataflow, labelled with the variables responsible for the flow.

2.4 Tools and environments

Existing research has shown us how to build open, integrated programming environments whose basic unit of composition is a programming language module. We are now working on the next step: providing environments that scale up to compositional systems. Toward this goal we are building on existing environment technology such as persistent object bases, tool integration mechanisms, and user-interface frameworks. To these we are adding capabilities such as those found in development environments that support constructing systems in terms of well-defined components and connectors. In particular we are extending current technology for generating specialized design environments that exploit the properties of specific compositional styles.

2.4.1 A taxonomy of architectures

We continued organizing and taxonomizing the architectural models that software developers use in practice [Garlan and Shaw 94c]. We presented a tutorial on this material at SIGSOFT in December. The tutorial was well attended and well-received by both university and industrial participants. We have been invited back for the next SIGSOFT, in late 1994.

2.4.2 Different notions of subtyping

The use of hierarchy is an important component of object-oriented design. Hierarchy allows the use of type families, in which higher level supertypes capture the behavior that all of their subtypes have in common. For this methodology to be effective, it is necessary to have a clear

understanding of how subtypes and supertypes are related. Our work takes the position that the relationship should ensure that any property proved about supertype objects also holds for its subtype objects. In [Liskov and Wing 93] we present two ways of defining the subtype relation, each of which meets this criterion, and each of which is easy for programmers to use. The subtype relation is based on the specifications of the sub- and supertypes; we present a way of specifying types that makes it convenient to define the subtype relation. We also discuss the ramifications of this notion of subtyping on the design of type families.

2.4.3 New compositional mechanisms for systems

Implicit invocation based on event announcement is an increasingly important technique for composing systems. However, the use of this technique has largely been confined to tool integration systems (in which tools exist as independent processes) and special-purpose languages (in which specialized forms of event broadcast are designed into the language from the start).

[Notkin et al. 93] broadens the class of systems that can benefit from this approach by showing how to augment general-purpose programming languages with facilities for implicit invocation.

2.4.4 Formalizing architectural connections

We consolidated our work on developing the Wright specification language. Wright now supports the formal description of architectural connectors as first class entities. We reported our work in [Allen and Garlan 94c, Allen and Garlan 94a].

2.4.5 Specifications in the Z language

[Jackson 94a] is our first and speculative step into automatic analysis of Z specifications. We laid the theoretical foundation for a technique that can check an infinite design by examining only a finite number of cases. We built a "baby" prototype that successfully checked several theorems from the literature, analyses previously performed only by hand.

[Jackson 95a] completes some specification work. We had previously shown how the notion of "projections" could be used to structure specifications in a new way that gives a better separation of concerns than existing forms of structuring, demonstrating practical utility by fully specifying a small (but real) Macintosh editor. We later simplified the ideas and worked out how they could benefit specifications written, in a more conventional manner, in the Z language.

2.4.6 Exploiting style in architectural design environments

To make an effective discipline of composable systems, it is becoming increasingly important to support architectural description and analysis with tools and environments. In [Garlan et al. 94b] we present a prototype system for developing architectural design environments that exploit architectural styles to guide software architects in producing specific systems. The primary technical innovations of this research were:

- A generic object model for representing architectural designs
- The characterization of architectural styles as specializations of this object model (through subclassing)
- A practical toolkit for creating an open architectural design environment from a description of a specific architectural style.

2.4.7 Signature matching for software libraries

Signature matching is a method for organizing, navigating through, and retrieving from software libraries. In [Zaremski and Wing 94] we consider two kinds of software library components—functions and modules—and, hence, two corresponding kinds of matching. The signature of a function is simply its type; the signature of a module is a multiset of user-defined types and a multiset of function signatures. For both functions and modules we consider not just exact match, but also various flavors of relaxed match. We describe various applications of signature matching as a tool for using software libraries, using results from our implementation of a function signature matcher written in Standard ML.

2.5 Software Architectures

Our accomplishments in the area of software architectures fall into several categories: reuse at the software architecture level, formalizing descriptions of software architectures, architectural styles (including making comparisons between them, and casting them into patterns for the patterns community), and the development of an interchange language to translate architectural descriptions from one architectural description language (ADL) to another.

Our research in software architectures evolved to include the following additional categories: architecture classification/taxonomy, interoperability and evolution, architecture description and analysis, reasoning about software architecture, research on architecture description languages (ADLs), development to support ADLs, specific examples and case studies, and design support in the form of architectural styles.

We have also made progress in the following research areas: making specifications of components into evolving documents (and acknowledging their incompleteness), and being able to define non-primitive connectors in an architectural description.

In addition, we have organized a model problem collection for the software architecture community to discuss, produced a summary of the conclusions from the First International Workshop on Architectures for Software Systems, and produced a work describing the challenging research problems in Software Architecture.

2.5.1 Reuse at the software architecture level

The first work below describes how the concept of "architectural style" is useful in supporting reuse at the software architecture level, and more specifically how the Aesop system's Software Shelf supports this. The second work describes some of the mismatch problems encountered while trying to reuse existing off-the-shelf software to compose a system.

Style-based reuse for software architecture

Although numerous mechanisms for promoting software reuse have been proposed and implemented over the years, most have focused on the reuse of implementation code. There is much conjecture and some empirical evidence, however, that the most effective forms of reuse are generally found at more abstract levels of software design. In [Monroe 96a] we discuss software reuse at the architectural level of design. Specifically, we argue that the concept of "architectural style" is useful for supporting the classification, storage, and retrieval of reusable

architectural design elements. We briefly describe the Aesop system's Software Shelf, a tool that assists designers in selecting appropriate design elements and patterns based on stylistic information and design constraints.

Architectural mismatch: why reuse is so hard

Many would argue that future breakthroughs in software productivity will depend on our ability to combine existing pieces of software to produce new applications. An important step towards this goal is the development of new techniques to detect and cope with mismatches in the assembled parts. Some problems of composition are due to low-level issues of interoperability, such as mismatches in programming languages or database schemas. However, in [Garlan 95c] we highlight a different, and in many ways more pervasive, class of problem: architectural mismatch. Specifically, we use our experience in building a family of software design environments from existing parts to illustrate a variety of types of mismatch that center around the assumptions a reusable part makes about the structure of the application in which it is to appear. Based on this experience we show how an architectural view of the mismatch problem exposes some fundamental, thorny problems for software composition and suggests possible research avenues needed to solve them. This paper is a revised version of a similar paper that appeared at ICSE-17.

2.5.2 Formalizing descriptions of software architecture

Formal methods offer a powerful approach to analyzing software architectures and architectural styles, and for describing the architecture of real systems. Our research addresses real world examples such as commercial electronic instruments and a military weapons system.

Formulations and formalisms in software architecture

Software architecture is the level of software design that addresses the overall structure and properties of software systems. It provides a focus for certain aspects of design and development that are not appropriately addressed within the constituent modules. Architectural design depends heavily on accurate specifications of subsystems and their interactions. These specifications must cover a wide variety of properties, so the specification notations and associated methods must be selected or developed to match the properties of interest. Unfortunately, the available formal methods are only a partial match for architectural needs, which entail description of structure, packaging, environmental assumptions, representation, and performance as well as functionality. Understanding what needs to be formalized is a prerequisite for devising or selecting a formal method. For software architecture, much of this understanding arises through progressive codification, which begins with real-world examples and creates progressively more precise models that eventually support formalization. In [Shaw 95d] we describe such an approach and explore the relation between emerging models and the selection, development, and use of formal systems.

Formalizing architectural style

The software architecture of most systems is usually described informally and diagrammatically by means of boxes and lines. For these descriptions to be meaningful, the diagrams are understood by interpreting the boxes and lines in specific, conventionalized ways. The informal, imprecise nature of these interpretations has a number of limitations. In [Abowd 96] we consider these conventionalized interpretations as architectural styles and provide a formal framework for

their uniform definition. In addition to providing a template for precisely defining new styles, this framework facilitates analysis within and between different existing styles.

Formal specification of an architecture for a family of systems

[Garlan and Delisle 95b] describes industrial experience developing a formal architectural model for a family of oscilloscopes. This paper is a revised version of a similar paper that appeared in IEEE Software, September 1990.

A case study in architectural modelling

Software architecture is receiving increasing attention as a critical design level for software systems. However, the current practice of architectural description is largely informal and ad hoc, with the consequence that architectural documents serve as a poor communication mechanisms, are difficult to analyze, and may have very little relationship to the implemented system. In an attempt to address these problems several researchers have experimented with formalisms for architectural specification and modelling. One such formalism is Wright. In [Allen 96a] we show how Wright can be used to provide insight into an architectural design by modelling a prototype implementation of part of the AEGIS Weapons System.

2.5.3 Architectural styles

A comparison of styles for software architecture

In [Shaw 95e] we examine a variety of different solutions to a specific design problem. The comparison reveals that each design relies on several models or architectural styles. We raise questions of (a) choosing the architecture that matches the problem at hand and (b) ensuring consistency of several overlapping views.

Some patterns for software architecture

Software designers rely on informal "patterns," or idioms, to describe the architectures of their software systems — the configurations of components that make up the systems. In [Shaw 95c] we establish a relationship with the patterns community by casting several popular architectural styles as patterns.

2.5.4 An interchange language for software architecture

A number of architectural description languages (ADLs) have been developed to support the formal representation and analysis of software architectures. Each such ADL provides certain complementary capabilities supporting architectural development. Unfortunately, however, each ADL and its supporting toolset typically operate in a stand-alone fashion, making it difficult to combine multiple tools, share architectural descriptions, or build on previous work in producing new languages and tools. ACME was developed as a joint effort of the architectural research community to provide a common interchange format for architectural design tools. ACME provides a structural framework for characterizing architectural designs together with liberal annotation facilities for including additional ADL-specific auxiliary information.

2.5.5 Software architecture primitives

The works below describe research regarding components and connectors. The first work addresses the way formalists think about specifications of components; the second describes abstractions for connectors, and the information required by a designer to implement a new abstraction in a specific architecture description language called UniCon.

The difference between what a component does and what we *know* it does

In [Shaw 96] we address the discrepancies between the way formalists think about specifications and the practical inability to anticipate and specify every property of a component that someone might depend on. We propose an approach to making specifications into evolving documents that capture what is currently known about a component.

Abstractions and implementations for architectural connections

The architecture of a software system shows how the system is constructed from components. The properties of the system depend critically on the character of the interactions among the components. Although software designers have good informal abstractions for these interactions, the abstractions are poorly supported by the available languages and tools. UniCon provides a rich selection of abstractions for the connectors that mediate interactions among components. To create systems using the connector abstractions, one needs to produce and integrate not only the object code for components, but also a variety of other run-time products. To extend the set of connectors supported by UniCon, one needs to identify and isolate many kinds of information in the compiler, graphical editor, and associated tools. In [Shaw et al. 96] we describe the role of connector abstractions in software design, the connector abstractions currently supported by UniCon, and implementation issues associated with supporting an open-ended collection of connectors. We describe progress toward being able to define non-primitive connectors.

2.5.6 Research conclusions and directions

The first work below summarize some important conclusions from the First International Workshop on software architecture. The last works describe a common set of problems around which the software architecture community might organize themselves to exhibit techniques and compare results, and describe the open problems in software architecture.

The first workshop on architectures for software systems

[Garlan 95d] summarizes the conclusions of the First International Workshop on Architectures for Software Systems, held in conjunction with the 17th International Conference on Software Engineering, April 1995.

2.5.7 Architecture classification/taxonomy

Toward boxology: preliminary classification of architectural styles

Software architects use a number of commonly-recognized "styles" to guide their design of system structures. Each of these is appropriate for some classes of problems, but none is suitable for all problems. How, then, does a software designer choose an architecture suitable for the problem at hand? Two kinds of information are required: (1) careful discrimination among the candidate architectures and (2) design guidance on how to make appropriate choices. In

[Clements 96] we support careful discrimination with preliminary classification of styles. We use a two-dimensional classification strategy with control and data issues as the dominant organizing axes. We position the major styles within this space and use finer-grained discriminations to elaborate variations on the styles. This provides a framework for organizing design guidance, which we partially flesh out with rules of thumb.

2.5.8 Architectural evolution

Conversion of batch systems to support interaction

Software often evolves from batch to interactive use. Because these two usage styles are so different, batch systems usually require substantial changes to support interactive use. Specific issues that arise during conversion include: assumptions about system execution time, incremental and partial processing, scope of processing, unordered and repeated processing, and error handling. Addressing these issues affects the implementation in the areas of memory management, assumptions and invariants, computational organization, and error handling. In [DeLine et al. 97] we present lessons for practitioners who are faced with similar types of conversions. We use as a working example our conversion of the batch processor for the UniCon architecture description language into an interactive architecture editor. We summarize with a checklist of design and implementation considerations.

2.5.9 Architecture description and analysis

ACME: An architecture description interchange language

Numerous architectural description languages (ADLs) have been developed, each providing complementary capabilities for architectural development and analysis. Unfortunately, each ADL and supporting toolset operates in isolation, making it difficult to integrate those tools and share architectural descriptions. ACME is being developed as a joint effort of the software architecture research community as a common interchange format for architecture design tools. ACME provides a structural framework for characterizing architectures, together with annotation facilities for additional ADL-specific information. This scheme permits subsets of ADL tools to share architectural information that is jointly understood, while tolerating the presence of information that falls outside their common vocabulary. In [Garlan 97a] we describe ACME's key features, rationale, and technical innovations.

Architectural styles, design patterns, and objects

Software system builders are increasingly recognizing the importance of exploiting design knowledge in the engineering of new systems. One way to do this is to define an architectural style for a collection of related systems. The style determines a coherent vocabulary of system design elements and rules for their composition. By structuring the design space for a family of related systems, a style can, in principle, drastically simplify the process of building a system, reduce costs of implementation through reusable infrastructure, and improve system integrity through style-specific analyses and checks.

Like architectural style, object-oriented design patterns attempt to capture and exploit design knowledge to ease the process of designing software systems and reusing proven designs. There

are, however, significant differences in the roles and capabilities of architectural styles and object-oriented design patterns, as there are between architectural design and object-oriented design. In [Monroe 97] we illustrate the relationship between software architecture and object-oriented design, as well as the relationship between architectural styles and design patterns.

Formal modeling and analysis of the HLA RTI

The High Level Architecture Run Time Infrastructure (HLA RTI) is a complex artifact, supporting several classes of interaction (e.g., federation management, object management, time management). A critical challenge in producing an RTI architectural framework (and its associated simulation interface specifications) is to develop confidence that its specification is well-formed and complete. In [Garlan 97b] we describe on-going work in formally modelling the HLA both to document the standard more precisely, and to analyze it for anomalies, omissions, inconsistencies, and ambiguities. The technical basis for this work is the use of a formal architectural description language, called Wright, and its accompanying toolset.

2.5.10 Reasoning about software architecture

Towards a formal treatment of implicit invocation

In [Dingel 97] we develop a formal basis for reasoning about certain classes of implicit invocation systems (sometimes called publish-subscribe systems). The work provides an operational and denotational semantics and illustrates how one can reason about the behavior of implicit invocation systems.

Architectural unification

Many software designs are produced by combining and elaborating existing architectural design fragments. These fragments may be design patterns, partially thought-out ideas, or portions of some previously-developed system design. To provide mechanized support for this activity it is necessary to have a precise characterization of when and how two or more architectural fragments can be combined. In [Melton 97] we describe extensions to notations for software architecture to represent incomplete design fragments, and algorithms for combining fragments in a process analogous to unification in logic.

Style-based refinement for software architecture

A question that frequently arises for architectural design is "When can I implement a design in style S1 using a design in style S2?". In [Garlan 96a] we propose a technique for structuring a solution to this kind of problem using the idea of substyles. This technique leads to a two-step process: First, useful subsets of a family of architectures are identified; and second, refinement rules specific to these subsets are established. We argue that this technique, in combination with an unconventional interpretation of refinement, clarifies how engineers actually carry out architectural refinement and provides a formal framework for establishing the correctness of those methods.

2.5.11 Architecture description languages

The difference between what a component does and what we *know* it does

Conventional doctrine holds that specifications are sufficient, complete, static, and homogeneous. For system-level specifications, especially for software architectures and their components, conventional doctrine often fails to hold. Specifications for real software must be incremental, extensible, and heterogeneous. To support such specifications, our notations and tools must be able to extend and manipulate structured specifications. In [Shaw 96] we propose an approach to making specifications into evolving documents that capture what is currently known about a component. The UniCon architecture description language introduces *credentials*, a property-list form of specification that supports evolving heterogeneous specifications and their use with system-building and analysis tools.

Abstractions and implementations for architectural connections

The architecture of a software system shows how the system is constructed from components. The system properties depend critically on the character of the component interactions. Although software designers have good informal abstractions for these interactions, the abstractions are poorly supported by the available languages and tools. UniCon provides a rich selection of abstractions for the connectors that mediate interactions among components. To create systems using the connector abstractions, one needs to produce and integrate not only the object code for components, but also a variety of other run-time products. To extend the set of connectors supported by UniCon, one needs to identify and isolate many kinds of information in the compiler, graphical editor, and associated tools. In [Shaw et al. 96] we describe the role of connector abstractions in software design, the connector abstractions currently supported by UniCon, and implementation issues associated with supporting an open-ended collection of connectors.

User-defined element types and architectural styles

When considering the design of an architecture description language (ADL) to be used as part of a software developer's daily practice, two goals merit attention. First, the language should support the easy definition of new element types and architectural styles. Second, it should play a central role in system construction. In [DeLine 96], we propose an ADL, called UniCon-2, that addresses these goals with its flexible type system, its duty construct, and its extensible compiler architecture based on OLE. This ADL provides a good starting point for exploring the architectural description of families of systems and flexible componentry.

2.5.12 Development to support ADLs

Style-based reuse for software architecture

Although numerous mechanisms for promoting software reuse have been proposed and implemented, most have focused on the reuse of implementation code. There is much conjecture and some empirical evidence, however, that the most effective forms of reuse are generally found at more abstract levels of software design. In [Monroe 96a] we discuss software reuse at the architectural level of design. Specifically, we argue that the concept of "architectural style" is useful for supporting the classification, storage, and retrieval of reusable architectural design elements. We briefly describe the Aesop system's Software Shelf, a tool that assists designers in

selecting appropriate design elements and patterns based on stylistic information and design constraints.

Capturing design expertise in customized software architecture design environments

Software architecture, with its own set of design issues, vocabulary, and goals, has emerged as a distinct form of abstraction for software systems. Like designers in other disciplines, software architects can gain significant leverage by using powerful design environments and tools. Such design tools generally encapsulate a relatively small amount of design expertise that provides the important functionality of the tool within a relatively large support infrastructure. In [Monroe 96b] we argue that, in order to make the development of specialized architectural design tools practical, capturing this design expertise so that it can be used to configure architectural design environments incrementally must be relatively easy and inexpensive.

2.5.13 Examples and case studies

A case study in architectural modelling

Software architecture is receiving increasing attention as a critical design level for software systems. However, the current practice of architectural description is largely informal and ad hoc, with the consequence that architectural documents serve as a poor communication mechanism, are difficult to analyze, and may have very little relationship to the implemented system. In an attempt to address these problems „several researchers have experimented with formalisms for architectural specification and modeling. One such formalism is *Wright*, and in [Allen 96a] we show how *Wright* can be used to provide insight into an architectural design by modelling a prototype implementation of part of the AEGIS Weapons System.

A standards effort as architectural style

In [Allen 96b] we introduce a case study, the DoD "High Level Architecture for Simulations (HLA)," and briefly discuss our efforts to apply *Wright* to the HLA. Our work on HLA has focused on understanding it as an architectural style and concentrates on the Interface Specification (IFSpec) description of the "Runtime Infrastructure (RTI)" as the central architectural design issue. Specifically, we have used *Wright* to characterize the RTI and analyze a number of its properties. By providing an analysis of the properties of the RTI as described by the IFSpec, we can help the standards committee to determine whether the IFSpec ensures the properties that they want and to discover inconsistencies or other weaknesses of the specification.

Stylized architecture, design patterns, and objects

Software system builders are increasingly recognizing the importance of exploiting design knowledge when engineering new systems. One way to do this is to define an architectural style for a collection of related systems. The style determines a coherent vocabulary of system design elements and rules for their composition. By structuring the design space for a family of related systems, a style can, in principle, drastically simplify the system building process, reduce implementation costs through reusable infrastructure, and improve system integrity through style-specific analyses and checks.

Like architectural style, object-oriented design patterns attempt to capture and exploit design knowledge to ease the process of designing software systems and reusing proven designs. There

are, however, significant differences in the roles and capabilities of architectural styles and object-oriented design patterns, as there are between architectural design and object-oriented design. In [Monroe 97] we illustrate the relationship between software architecture and object-oriented design, as well as the relationship between architectural styles and design patterns.

Stylized architecture, design patterns, and objects

The software architecture and the design pattern communities have overlapping interests. The software architecture community is chiefly concerned with structure and organization of large software systems; the patterns community with exposition of design information. These intersect in the exposition of design information at the system level. [Shaw et al 96] lays out a framework for the software engineering community to consider (a) what new ADL capability is suggested by design patterns and (b) to what extent ADLs are appropriately carriers of pattern information, and how they should do so.

2.5.14 Design support: architectural styles

An architectural style can be viewed as a coherent set of constraints on software design. In this section we describe our research pertaining to architectural styles.

Preliminary classification of architectural styles

Software architects use a number of commonly-recognized "styles" to guide their design of system structures. In [Clements 96] we begin to classify these styles. We use a two-dimensional classification strategy with control and data issues as the dominant organizing axes. We position the major styles within this space and use finer-grained discriminations to elaborate variations on the styles. This provides a framework for organizing design guidance.

Style-based refinement for software architecture

A question that frequently arises for architectural design is "When can I implement a design in style S1 using a design in style S2?" In [Garlan 96a] we propose a technique for structuring a solution to this kind of problem using the idea of sub-styles. This technique leads to a two-step process in which, first, useful subsets of a family of architectures are identified and, second, refinement rules specific to these subsets are established. We argue that this technique, in combination with an unconventional interpretation of refinement, clarifies how engineers actually carry out architectural refinement and provides a formal framework for establishing the correctness of those methods.

2.5.15 The software architecture discipline

Software architecture: perspectives on an emerging discipline

In [Shaw and Garlan 96] we examine architectures for software systems as well as better ways to support software development. We attempt to bring together the useful abstractions of system design and the notations and tools of the software developer, and look at patterns used for system organization.

This book presents an introduction to the field of software architecture. Our purpose is to illustrate the discipline and examine the ways in which architectural design can impact software

design. Our selection emphasizes informal descriptions, touching lightly on formal notations and specifications and on tools to support them.

Our book provides a suitable text for a course on software system architectures. It brings together the emerging models for software architectures and shows how to approach systems from an architectural point of view.

2.6 Formal methods

Our accomplishments in the area of formal methods fall into three categories: structuring Z specifications, formal specification of concurrent systems, and specification matching of software components.

2.6.1 Structuring Z specifications with views

In [Jackson 95a] we explain an approach to specification structuring that gives a better separation of concerns than conventional approaches. We explain the notion of "views," give examples of views and their composition, articulate the language features that view structuring relies upon, and suggest some open research problems in language design and specification methodology.

2.6.2 Formal specification of concurrent systems

In [Chadha et al. 95] we present a formal methodology for developing concurrent systems. We extend the Larch family of specification languages and tools with the CCS process algebra to support the specification and verification of concurrent systems. We present and follow a refinement strategy that relates an implementation in a programming language to a formal specification of such a system. We illustrate our methodology on an example that uses the preconditioned conjugate gradient method for solving a linear system of equations.

2.6.3 Specification matching of software components

Specification matching is a way to compare two software components. In the context of software reuse and library retrieval, it can help determine whether one component can be substituted for another or how one can be modified to fit the requirements of the other. In the context of object-oriented programming, it can help determine when one type is a behavioral subtype of another. In the context of system interoperability, it can help determine whether the interfaces of two components mismatch.

In [Zaremski and Wing 95] we use formal specifications to describe the behavior of software components and, hence, to determine whether two components match. We give precise definitions of exact match and, more relevantly, various flavors of *relaxed* match. These definitions capture the notions of generalization, specialization, substitutability, subtyping, and interoperability of software components.

We write our formal specifications of components in terms of pre- and post-condition predicates. Thus, we rely on theorem proving to determine match and mismatch. We give examples from our implementation of specification matching using the Larch Prover.

This work describes a part of the larger body of research done by Amy Zaremski, a PhD candidate in the School of Computer Science, in her thesis [Zaremski 96]. That work defines the foundations for signature and specification matching. It defines matching in a general, extensible framework; how matching can be applied to the problems of retrieval from libraries, indexing libraries, and reuse of components; and provides implementations that demonstrate the feasibility of the approach and illustrate the usefulness of the applications with results from a moderately-sized component library.

In [Zaremski and Wing 97a] we use formal specifications to describe the behavior of software components, and hence, to determine whether two components match. We give precise definitions of not just exact match, but more relevantly, various flavors of relaxed match. These definitions capture the notions of generalization, specialization, and substitutability of software components.

Since our formal specifications are pre- and post-conditions written as predicates in first-order logic, we rely on theorem proving to determine match and mismatch. We give examples from our implementation of specification matching using the Larch Prover.

2.6.4 Protective interface specifications

The interface specification of a procedure describes the procedure's behavior using pre- and postconditions. These pre- and postconditions are written using various functions. If some of these functions are partial, or underspecified, then the procedure specification may not be well-defined.

In [Leavens and Wing 97] we show how to write pre- and postcondition specifications that avoid such problems, by having the precondition "protect" the postcondition from the effects of partiality and underspecification. We formalize the notion of protection from partiality in the context of specification languages like VDM-SL and COLD-K. We also formalize the notion of protection from underspecification for the Larch family of specification languages, and (for Larch) show how one can prove that a procedure specification is protected from the effects of underspecification.

2.6.5 Formal methods: state of the art and future directions

In [Clarke 96] we survey recent progress in the development of mathematical techniques for specifying and verifying complex hardware and software systems. Many of these techniques are capable of handling industrial-sized examples; in fact, in some cases these techniques are already being used on a regular basis in industry. Success in formal specification can be attributed to notations that are accessible to system designers and to new methodologies for applying these notations effectively. Success in verification can be attributed to the development of new tools such as more powerful theorem provers and model checkers than were previously available. Finally, we suggest some general research directions that we believe are likely to lead to technological advances. Although it is difficult to predict where the future advances will come from, optimism about the next generation of formal methods is justified in view of the progress during the past decade. Such progress, however, will depend heavily on continued support for basic research on new specification languages and new verification techniques.

2.6.6 Hints to specifiers

In [Wing 96] we present a list of hints for writing specifications. We address high-level issues like learning to abstract and low-level issues like getting the details of logical expressions right. This work should be of interest not only to students of formal methods but also to their teachers.

A major goal of software engineering is to enable developers to construct systems that operate reliably despite increasing complexity. One way of achieving this goal is by using formal methods, which are mathematically-based languages, techniques, and tools for specifying and verifying such systems. In this area, the composable systems group has been doing research in a number of different areas. First, we are using formal specifications to describe the behavior of software components to determine if two components match, or if one is a behavioral subtype of another. Second, we are investigating lightweight approaches to formal methods to make analysis of specifications economically feasible. A lightweight approach, which emphasizes partiality and focused application, can bring some of the benefits of formal methods to bear at reduced cost. Lastly, we are doing research in underspecification. Specifically, we illustrate techniques for ensuring that implementations do not become dependent on parts of specifications that are intentionally underspecified. The composable systems group also compiled a report on the state of the art of formal methods and their future directions.

2.6.7 Specification matching of software components

Specification matching is a way to compare two software components based on descriptions of the components' behaviors. In the context of software reuse and library retrieval, it can help determine whether one component can be substituted for another or how one can be modified to fit the requirements of the other. In the context of object-oriented programming, it can help determine when one type is a behavioral subtype of another.

2.6.8 Lightweight formal methods

For everyday software development, the purpose of formalization is to reduce the risk of serious specification and design errors. Analysis can expose such errors while they are still cheap to fix. Formal methods can provide limited guarantees of correctness too, but, except in safety-critical work, the cost of full verification is prohibitive and early detection of errors is a more realistic goal.

To make analysis economically feasible, the cost of specification must be dramatically reduced, and the analysis itself must be automated. Experience (of several decades) with interactive theorem proving has shown that the cost of proof is usually an order of magnitude greater than the cost of specification. And yet the cost of specification alone is often beyond a project's budget. Industry will have no reason to adopt formal methods until the benefits of formalization can be obtained immediately, with an analysis that does not require further massive investment.

Existing formal methods, at least if used in the conventional manner, cannot achieve these goals. By promoting full formalization in very expressive languages, they have guaranteed that the benefits of formalization are spread very thin. A lightweight approach, which, in contrast, emphasizes partiality and focused application, can bring greater benefits at reduced cost. In [Jackson and Wing 96] we present the elements of a lightweight approach to the use of formal methods.

A lightweight approach, in comparison to the traditional approach, lacks power of expression and breadth of coverage. A surgical laser likewise produces less power and poorer coverage than a common light bulb, but it makes more efficient use of the energy it consumes, and its effect is more dramatic.

2.6.9 Formal methods: state of the art and future directions

Hardware and software systems will inevitably grow in scale and functionality. Because of this increase in complexity, the likelihood of subtle errors is much greater. Moreover, some of these errors may cause catastrophic loss of money, time, or even human life. While the use of formal methods does not a priori guarantee correctness, they can greatly increase our understanding of a system by revealing inconsistencies, ambiguities, and incompletenesses that might otherwise go undetected.

In [Clarke 96] we assess the state of the art in specification and verification. For verification, we highlight advances in model checking and theorem proving. In the three sections on specification, model checking, and theorem proving, we explain what we mean by the general technique and briefly describe some successful case studies and well-known tools. We outline future directions in fundamental concepts, new methods and tools, integration of methods, and education and technology transfer. We close with summary remarks and pointers to resources for more information.

2.6.10 Protection from the underspecified

Underspecification is a good way to deal with partial functions in specification and reasoning. However, when underspecification is used, implementations may unintentionally be forced to depend on parts of the specification that were supposed to be underspecified. In [Leavens and Wing 96] we show how to write pre- and post-condition specifications that avoid such problems, by having the precondition "protect" the postcondition from the effects of underspecification. This approach is most practical if the specification of mathematical vocabulary is separated from the specification of implementation behavior, as in Larch, because it gives the specifier a chance to think about protection separately from the specification of mathematical behavior. We formalize the notion of protective procedure specifications, and show how to prove that a specification is protective. We also extend the Larch Shared Language to allow specification of what is intentionally left underspecified, permitting enhanced debugging of such specifications.

2.7 Tractable software analysis

Our accomplishments in the area of software analysis fall into four categories: model checking of software systems using SMV, Nitpick, and FDR (Failure-Divergence-Refinement), and analysis of software systems.

Our research in software analysis later evolved to include: new analysis mechanisms, analysis and model checking case studies, a new approach to verifying properties of security protocols called "theory checking," and a new method (and tool, called Lackwit) for computing representation sharing in a program.

2.7.1 Formalizing the uni-processor simplex architecture

Simplex is a software architecture developed by the Software Engineering Institute for dependable and evolvable process-control systems. In [Rivera and Danylyszyn 95] we describe our work which consisted of creating a formal specification of the Simplex architecture and analyzing its safety and liveness properties.

We developed a CSP model to describe the overall dynamic behavior of the Simplex architecture. We then verified the model using the Failure-Divergence-Refinement (FDR) model checker.

We also developed a Wright specification of this architecture to characterize precisely the connections between its components at the architectural level. The specification was based on the latest version of the CSP model.

2.7.2 Analysis and model checking case studies

The following works describe the Nitpick specification checker, the NP specification language, three reduction mechanisms employed in the checker to reduce the search space of a problem (and thereby speed up the checking, allowing for the checking of much larger specifications), and a case study in using the checker on a real system.

Nitpick release

This year we have developed a new technology for automatic analysis of software specifications. We have shown that the checking and simulated execution of specifications written in conventional software specification languages such as Z can be reduced to the problem of generating finite models of a relational formula.

We built the Nitpick Checker, a model generator for a subset of Z called NP. The NP language is designed to tradeoff expressiveness against tractability; it retains the most powerful specification constructs of Z, eliminates some features that may hinder analysis, and incorporates the structuring mechanisms of Z (the schema calculus) in a simplified and improved form.

The checker employs three reduction mechanisms (explained in our papers, see below) that are capable of reducing the search for models by huge factors. We are able to analyze spaces of up to 10^{23} cases, for example, by reduction to a couple of million.

A beta version of the Nitpick Checker has been released and is currently being used in a masters' level software engineering course on software analysis.

Nitpick: A checkable specification language

[Jackson 96a] outlines the design rationale behind the NP specification language.

In [Damon 96] we explain the short-circuiting mechanism of Nitpick, a method for reducing the search for models by pruning the search. A model is constructed incrementally; by showing that no extension of a partial model can be a satisfying model, huge search reductions can be obtained. Short-circuiting shows great promise: It gives larger reductions the more complex the property being checked, and has no time or space overhead except in preprocessing.

[Jackson 96b] explains the isomorph reduction mechanism. Nitpick exploits both the structure of the specification being checked and the structure of the elements of the models to reduce the search for models. By avoiding generating isomorphs, reductions of 10^5 are easily obtained; as the size of the models to be generated increases, the isomorph reduction increases exponentially.

Nitpick's derived variable reduction is explained in [Jackson 95b]. By static analysis, Nitpick can discover constructiveness in the specification of an operation, allowing the value of a variable of the post-state to be derived directly from a value of a variable in the pre-state, eliminating the need for independent search. Often half the variables in a specification can be eliminated in this fashion.

Analyzing a software design feature with a counterexample detector

In [Jackson 96c] we present a case study that applies Nitpick to a small but realistic specification. We characterize the behavior of the paragraph style mechanism in Microsoft Word, and show, by the generation of a series of models refuting expected properties, that the mechanism is fundamentally flawed.

We illustrate the application of a checking tool to the design of a style mechanism for a word processor. The design is cast, along with some expected properties, in a subset of Z that corresponds to the relational calculus. The tool evaluates a property by enumerating all possible cases within some finite bounds, displaying as a counterexample the first case for which the property fails to hold. Unlike animation or execution tools, our checker does not require state transitions to be expressed constructively, and unlike theorem provers, operates completely automatically without user intervention. Using a variety of reduction mechanisms, it can cover an enormous number of cases in a reasonable time, so that quite subtle flaws can be rapidly detected.

Automatic analysis of architectural style

In [Jackson 96d] we present a case study application of Nitpick to architectural styles. We show how Nitpick can expose a variety of undesirable properties that might arise in instances of an architectural style. Several variants of implicit invocation systems (originally formalized by Garlan and Notkin) are analyzed; the checker generated particular architectures that satisfy the style rules but may still exhibit bad behaviors, such as races and cycles.

2.7.3 Model checking software systems: A case study

Model checking is a proven technology for verifying hardware. It works, however, only on finite state machines, and most *software* systems have infinitely many states. Our approach to applying model checking to software hinges on identifying appropriate abstractions that exploit the nature of both the system, S , and the property, Φ , to be verified. We check Φ on an abstracted, but finite, model of S .

Following this approach we verified three cache-coherence protocols used in distributed file systems [Wing and Vaziri-Farahani 95]. These protocols must satisfy the following property: "If a client believes that a cached file is valid, then the authorized server believes that the client's copy is valid." In our finite model of the system, we need only represent the "beliefs" that a client and a server have about a cached file; we can abstract from the caches, the files' contents, and even the files themselves. Moreover, by successive application of the generaliza-

tion rule from predicate logic, we need only consider a model with, at most, two clients, one server, and one file. We used McMillan's SMV model checker: On our most complicated protocol, SMV took less than 1 second to check over 43,600 reachable states.

We also include a description of the verification of the Coda protocol in a more complete version of the work [Wing and Vaziri-Farahani 96].

Model checking electronic commerce protocols

In [Heintze et al. 96] we develop model checking techniques to examine NetBill and Digicash. We show how model checking can find atomicity problems by analyzing simplified versions of these protocols that retain crucial security problems. For our analysis we used the FDR model checker.

Additional Nitpick developments

We developed a new symmetry reduction method for our Nitpick specification checker. In the summer of 1996, we began work on a simpler proof of soundness of our existing method; while working on the proof, we found a way to improve the algorithm. The new algorithm was incorporated into the Nitpick tool in the fall of 1997; it is not only cleaner but also performs significantly better.

We have been researching new boolean-based specification checking and bounded generation techniques for our Nitpick tool. Nitpick checks properties of software specifications by exhaustive enumeration of states. For complex states involving many relations, functions and sets, the state space is vast. The current Nitpick tool employs a variety of mechanisms to reduce the space. The first is symmetry reduction; it avoids considering a state that is symmetrical to one already analyzed. The second is short circuiting; it avoids enumerating the values of a particular state variable when it can be determined, from the values of other variables already assigned, that the property will be satisfied however the state assignment is completed. Together, these have enabled us to check some complex specifications, but the method is still very sensitive to the number of variables. Short circuiting essentially provides a weak form of goal-directedness. Values of variables are considered in their entirety - that is, a whole function, or a whole set. If it were possible to consider partial values, the goal-directedness could be much improved. We have started to work on a new checking engine that considers the arcs of a function or relation one at a time, so that the search has a finer granularity. Our scheme involves translating the relational specification into a boolean formula and applying satisfiability algorithms. It builds on our previous work, and exploits some of the ideas from our symmetry work. We have also been working on a scheme that improves the explicit search method. "Bounded generation" allows only the values of a relation to be generated that are known in advance to satisfy a constraint. For example, given a constraint that p is a sub-relation of q , and given that q has already been generated, the generation of p can be limited to sub-relations of q . A prototype implementation of bounded generation has been constructed, and appears to result in considerable speedup for a variety of specifications.

We used Nitpick to verify two properties of the Mobile IPv6 protocol (adopted as an IETF standard). The properties we checked are that the cache entries of mobile hosts do not form a cycle (so packets never travel in a loop) and that authenticated messages do not form a cycle. We found an error in the Mobile IPv6 protocol specification where a cycle, involving just two hosts,

could be created. We informed one of the designers of the problem, since IPv4 does not have the same error. We also found that a feature that was originally included as a performance optimization was necessary for the correctness of the protocol.

2.7.4 Theory checking

Fast, automatic checking of security protocols

Protocols in electronic commerce and other security-sensitive applications require careful reasoning to demonstrate their robustness against attacks. Several logics have been developed for doing this reasoning formally, but protocol designers usually do the proofs by hand, a process which is time-consuming and error-prone.

In [Kindred 96] we present a new approach, theory checking, to analyzing and verifying properties of security protocols. In this approach we generate the entire finite theory, T_η , of a logic for reasoning about a security protocol; determining whether it satisfies a property, π_η , is thus a simple membership test: π_η in T_η . Our approach relies on (1) modeling a finite instance of a protocol in the way that the security community naturally, though informally, presents a security protocol, and (2) placing restrictions on a logic's rules of inference to guarantee that our algorithm terminates, generating a finite theory. A novel benefit to our approach is that because of these restrictions we can provide an automatic theory-checker generator. We applied our approach and our theory-checker generator to three different logics for reasoning about authentication and electronic commerce protocols: the Burrows-Abadi-Needham logic of authentication, AUTLOG, and Kailar's accountability logic. For each, we verified the desired properties using specialized theory checkers; most checks took less than two minutes, the longest less than fifteen minutes.

2.7.5 Program understanding

A program-understanding tool based on type inference

Many questions that arise in reverse engineering or restructuring a program can be answered by determining, statically, where the structure of the program requires sets of variables to share a common representation. With this information we can find abstract data types, detect abstraction violations, identify unused variables, functions, and fields of data structures, detect simple errors of operations on abstract data types (such as failure to close after open), and locate sites of possible references to a value.

We have developed a method for computing representation sharing by using types to encode representations. We use polymorphic type inference to compute new types for all variables, eliminating cases of incidental type sharing where the variables might have different representations. The method is fully automatic and smoothly integrates pointer aliasing and higher-order functions. Because it is fully modular and computationally inexpensive, it should scale to very large systems.

In [O'Callahan 97] we report on our progress with the Lackwit tool. It now has a coherent formal foundation in terms of type theory and is proving remarkably effective in the analysis of large C programs. It gives information of a depth that is unprecedented in tools that can handle

systems of more than a few thousand lines of code, and it smoothly incorporates troublesome language features such as aliasing and higher-order functions. We are now applying Lackwit to the entire Linux operating system, and it appears that it will scale effectively to over 100K lines of code, producing accurate and useful results that could not be obtained by other tools. The results of an analysis provided by Lackwit can help to answer questions arising from reverse engineering or restructuring of code in programs.

2.7.6 Analysis of software systems

The following works describe new techniques for analyzing software systems. The first describes an alternative to program slicing for extracting data flow information at the system level. The second one describes a technique for decomposing problems into fragments that have conventional solutions.

Detecting shared representations using type inference

In [O'Callahan 95] we explain a new approach we are investigating to analyze large systems. By applying type inference techniques (of the sort used in the checking of expressions in the functional language ML) to C programs, we are able to extract data flow information at the system level far more cheaply than by existing techniques, while considering aliasing and pointer structures.

We are now beginning to develop a prototype reverse engineering tool embodying these ideas.

Problem decomposition for reuse

In [Jackson and Jackson 96] we apply view structuring to a model problem in software specification. We show how views can help decompose the problem into fragments that are amenable to rote solutions, because they correspond to the components of "problem frames," elements of problems that are well understood and have conventional and straightforward solutions.

2.7.7 New analysis mechanisms

Efficient search as a means of executing specifications

In [Damon 96] we explain the short-circuiting mechanism of Nitpick, a method for reducing the search for models by pruning the search. A model is constructed incrementally; by showing that no extension of a partial model can be a satisfying model, huge search reductions can be obtained. Short-circuiting shows great promise: it gives larger reductions the more complex the property being checked, and has no time or space overhead except in preprocessing.

Checking relational specifications with binary decision diagrams

Checking a specification in a language based on sets and relations (such as Z) can be reduced to the problem of finding satisfying assignments, or models, of a relational formula. In [Damon, Jackson, and Jha 96] we present a new method for finding models using ordered binary decision diagrams (BDDs). This method appears to scale better than existing methods.

Relational terms are replaced by matrices of Boolean formulae. These formulae are then composed to give a Boolean translation of the entire relational formula. Throughout, Boolean for-

mulae are represented with BDDs; from the resulting BDD, models are easily extracted. The performance of the BDD method is compared to our previous method based instead on explicit enumeration. The new method performs as well or better on most of our examples, but can also handle specifications that, until now, we have been unable to analyze.

Isomorph-free model enumeration: a new method for checking relational specifications

In [Jackson 97] we report a new algorithm for checking abstract software specifications. It is an advance on our previous work [Jackson 96b], being both simpler to explain and implement as well as more powerful — that is, resulting in a more efficient search. The algorithm has been implemented in the latest version of the Nitpick specification checker.

Checking relational specifications with binary decision diagrams

[Damon, Jackson, and Jha 96] reports an experiment in using boolean techniques in the engine at the heart of the Nitpick checker, in contrast to the techniques based on isomorph elimination, on which we have concentrated primarily. We used Bryant's ordered binary decision diagrams and were able to analyze specifications that we were previously unable to analyze. We plan to pursue this work more generally by looking at boolean satisfaction algorithms as a tool for specification checking.

2.7.8 Software engineering

Lessons on converting batch systems to support interaction

Software often evolves from batch to interactive use. Because these two usage styles are so different, batch systems usually require substantial changes to support interactive use. In [DeLine et al. 97] we enumerate specific issues that arise during such a conversion. These issues include assumptions about system execution duration, incremental and partial processing, scope of processing, unordered and repeated processing, and error handling. We discuss how addressing these issues affects the implementation in the areas of memory management, assumptions and invariants, computational organization, and error handling. We use as a working example our conversion of the batch processor for the UniCon architecture description language into an interactive architecture editor. To capture the lessons for practitioners undertaking this type of conversion, we summarize with a checklist of design and implementation considerations.

Scalable object oriented techniques for information integration in heterogeneous information systems

Many applications require information produced by outside parties; however, information from collections such as the World Wide Web is diverse enough in form to make information integration difficult. The object-oriented model gives a standard interface for access to information, but most OO systems can only manipulate objects specially created for that system.

A well-designed object model, backed up by a network of mediator agents, can apply object-oriented abstractions to a wide range of data, including Web documents. This encapsulation allows programs to exploit the structure of data types on the Web, including unfamiliar data types, and also to extend the types scalably.

2.8 Other research

The following sections summarize research in Composable Software Systems performed by two PhD candidates.

2.8.1 Safe and efficient persistent heaps

Object-oriented databases, persistent programming languages and distributed object servers are just some of the systems that require support for making arbitrary types of data persistent. Persistent heaps are essential components of such systems. The data stored in persistent heaps are valuable and hard to recreate, so it is crucial that this data be protected from both machine failures and programmer errors. This safety requirement may conflict with the need to provide high-throughput, low-latency access to the data, leading to a sacrifice of safety for performance.

In [Nettles 95] we discuss the design, implementation and performance evaluation of the first system that avoids the need to sacrifice safety for performance in persistent heaps.

The design uses an approach based on transactions and garbage collection to provide safe management of persistent data. Good performance is achieved by combining traditional systems techniques for transactions with a novel concurrent garbage collection technique, "replicating collection."

The implementation is the first to provide concurrent collection of a transactional heap. Replicating collection allows a much simpler implementation than previous (unimplemented) designs based on earlier concurrent collection techniques. The implementation is an extension of the runtime system of Standard ML of New Jersey.

2.8.2 Language support for mobile agents

Mobile agents are code-containing objects that may be transmitted between communicating participants in a distributed system. As opposed to systems that only allow the exchange of non-executable data, systems incorporating mobile agents can achieve significant gains in performance and functionality.

A programming language for mobile agents must be able to express their construction, transmission receipt, and subsequent execution. Its implementation must handle architectural heterogeneity between communicating machines and provide sufficient performance for applications based on agents. In addition to these *essential properties*, an agent language may support *desirable properties* such as high-level abstractions for code manipulation and the ability to access resources on remote execution sites. In [Knabe 95] we describe our language support for mobile agents.

We designed and implemented an agent-programming language that satisfies the essential properties and a number of desirable ones. A key feature of our language is the use of strong static typing for remote resource access. Agents may be linked dynamically to resources on remote sites, and this linking is always guaranteed to be type safe. We provide this guarantee without requiring that all components of an agent-based system be compiled together.

2.9 Bibliography

[Abowd 96]

Abowd, G., Allen, R., and Garlan, D.

Formalizing Style to Understand Descriptions of Software Architecture.

ACM Transactions on Software Engineering and Methodology, 1996.

[Abowd et al. 93]

Abowd, G., R. Allen, and D. Garlan.

Using style to give meaning to software architecture.

In *Proceedings of SIGSOFT'93: Foundations of Software Engineering*. December, 1993.

[Abowd et al. 95]

Abowd, G., R. Allen, and D. Garlan.

Formalizing Architectural Style.

IEEE Transactions on Software Engineering and Methodology, 1995.

[Allen 96a]

Allen, R. and Garlan, D.

A Case Study in Architectural Modelling: The AEGIS System.

In *Proceedings of the Eighth International Workshop on Software Specification and Design (IWSSD-8)*. Paderborn, Germany, March, 1996.

[Allen 96b]

Allen, R.

HLA: A Standards Effort as Architectural Style.

In *Proceedings of the Second International Software Architecture Workshop*. Second International Software Architecture Workshop, ACM Press, San Francisco, CA, October, 1996.

[Allen 97]

Allen, R.

A Formal Approach to Software Architecture.

PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1997.

Available as technical report CMU-CS-97-144.

[Allen and Garlan 94a]

Allen, R. and D. Garlan.

Formalizing architectural connection.

In *Proceedings of the Sixteenth International Conference on Software Engineering*. ICSE, May, 1994.

[Allen and Garlan 94b]

Allen, R. and D. Garlan.

Beyond definition/use: architectural interconnection.

In *Proceedings of the ACM Workshop on Interface Definition Languages*. ACM, January, 1994.

[Allen and Garlan 94c]

Allen, R. and D. Garlan.

A formal basis for architectural connection.

ACM Transactions on Software Engineering and Methodology, 1994.

[Allen and Garlan 97a]

Allen, R. and D. Garlan.

Formal Modeling and Analysis of the HLA RTI.

In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*. March, 1997.

[Allen and Garlan 97b]

Allen, R. and D. Garlan.

A Formal Basis for Architectural Connection.

Transactions on Software Engineering and Methodology, July, 1997.

To appear.

[Brown 97]

Brown, R.D.

Automated Dictionary Extraction for "Knowledge-Free" Example-Based Translation.

In *Proceedings of the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation*. July, 1997.

Santa Fe, NM.

An Example-Based Machine Translation system is supplied with a sentence-aligned bilingual corpus, but no other knowledge sources. Using the knowledge implicit in the corpus, it generates a bilingual word-for-word dictionary for alignment during translation. With such an automatically-generated dictionary, the system covers (with equivalent quality) *more* of its input on unseen texts than the same system does when provided with a manually-created general-purpose dictionary *and* other knowledge sources.

[Chadha et al. 94]

Chadha, H.S., J.W. Baugh, Jr. and J.M. Wing.

Formal specification of AEC product models.

In *Computing in Civil Engineering: Proceedings of the First Congress*. ASCE, June, 1994.

[Chadha et al. 95]

Chadha, H.S., Baugh, J.W., Jr., and Wing, J.M.

Formal Specification of Concurrent Systems.

Engineering with Computers, September, 1995.

Submitted.

[Clarke 96]

Clarke, E.M., Wing, J.M.

Formal Methods: State of the Art and Future Directions.

In *ACM Computing Surveys*. December, 1996.

Also in *Draft report of the Formal Methods Working Group, ACM Workshop on Strategic Directions in Computing Research*, Aug, 1996; and available as Carnegie Mellon University technical report CMU-CS-96-178.

[Clarke and Wing 96]

Clarke, E.M. and J. Wing.

Formal Methods: State of the Art and Future Directions.

ACM Computing Surveys 28(4):626-643, December, 1996.

[Clements 96]

Clements, P., Shaw, M.

Toward Boxology: Preliminary Classification of Architectural Styles.

In *Proceedings of the Second International Software Architecture Workshop*. ISAW, October, 1996.

[Damon 96]

Damon, C.A. and Jackson, D.

Efficient Search as a Means of Executing Specifications.

In *Proceedings of the Conference on Tools for Construction and Analysis of Software*. March, 1996.

[Damon, Jackson, and Jha 96]

Damon, C., D. Jackson, and S. Jha.

Checking Relational Specifications with Binary Decision Diagrams.

In *Proceedings of the 4th ACM SIGSOFT Conference on Foundations of Software Engineering*, pages 70-80. ACM, October, 1996.
San Francisco.

[DeLine 96]

DeLine, R.

Toward User-Defined Element Types and Architectural Styles.

In *Proceedings of the Second International Software Architecture Workshop*. ISAW, ACM Press, San Francisco, CA, October, 1996.

[DeLine et al. 97]

DeLine, R., Shaw, M., Zelesnik, G.

Lessons on Converting Batch Systems to Support Interaction.

In *Proceedings of the Eighteenth International Conference on Software Engineering*. 1997
ICSE, May, 1997.
To appear.

[DeLine, Zelesnik, and Shaw 97]

DeLine, R., G. Zelesnik, and M. Shaw.

Lessons on Converting Batch Systems to Support Interaction.

In *Proceedings of the 18th International Conference on Software Engineering*, pages 195-204.
May, 1997.

[Dingel 97]

Dingel, J., Garlan, D., Jha, S., Notkin, D.

Towards a Formal Treatment of Implicit Invocation.

January, 1997.

Submitted.

[Garlan 94]

Garlan, D.

Using refinement to understand architectural connection.

In *Proceedings of the Sixth Refinement Workshop*. January, 1994.

[Garlan 95a]

Garlan, D.

Research Directions in Software Architecture.

ACM Computing Surveys 27(2), June, 1995.

[Garlan 95b]

Garlan, D.

What is style?

In *Proceedings of the First International Workshop on Software Architecture*. April, 1995.

[Garlan 95c]

Garlan, D., Allen, R., and Ockerbloom, J.
Architectural Mismatch: Why Reuse is So Hard.
IEEE Software 12(6):17-28, November, 1995.

[Garlan 95d]

Garlan, D.
First International Workshop on Architectures for Software Systems: Workshop Summary.
ACM Software Engineering Notes:84-89, July, 1995.

[Garlan 96a]

Garlan, D.
Style-Based Refinement for Software Architecture.
In *Proceedings of the Second International Software Architecture Workshop*. ISAW, ACM Press, San Francisco, CA, October, 1996.

[Garlan 96b]

Garlan, D.
Style-Based Refinement for Software Architecture.
In *Proceedings of the Second International Software Architecture Workshop (ISAW2)*. October, 1996.

[Garlan 97a]

Garlan, D., Monroe, R.T., and Wile, D.
ACME: An Architecture Description Interchange Language.
January, 1997.
Submitted.

[Garlan 97b]

Allen, R., Garlan, D.
Formal Modeling and Analysis of the HLA RTI.
In *Spring Simulation Interoperability Workshop, Orlando, FL*. SIW, March, 1997.
To appear.

[Garlan and Delisle 95a]

Garlan, D. and N. Delisle.
Formal Specification of an Architecture for a Family of Instrumentation Systems.
Applications of Formal Methods.
In Hinchey and Bowen,
Prentice Hall, International Series in Computer Science, 1995.

[Garlan and Delisle 95b]

Garlan, D. and Delisle, N.
Formal Specification of an Architecture for a Family of Instrumentation Systems.
In Michael G. Hinchey and Jonathan P. Bowen (editor), *Applications of Formal Methods*, chapter 4. Prentice Hall, International Series in Computer Science, Hemel Hempstead, 1995.

[Garlan and Perry 95a]

Garlan, D. and D. Perry.
Introduction to the Special Issue on Software Architecture.
IEEE Transactions on Software Engineering and Methodology 21(4), 1995.

- [Garlan and Perry 95b]
Garlan, D. and D. Perry.
Software Architecture: Practice, Potential, and Pitfalls.
In Proceedings of the Sixteenth International Conference on Software Engineering. ICSE,
May, 1995.
- [Garlan and Shaw 94a]
Garlan, D. and M. Shaw.
Software development assignments for a software architecture course.
In Proceedings of the Sixteenth International Conference on Software Engineering. ICSE,
May, 1994.
- [Garlan and Shaw 94b]
Garlan, D. and M. Shaw.
Programming exercises for software architecture.
In ICSE-16 workshop on software engineering education. 1994.
- [Garlan and Shaw 94c]
Garlan, D. and M. Shaw.
An introduction to software architecture.
Advances in Software Engineering and Knowledge Engineering, Volume I.
World Scientific Publishing Company, 1994.
- [Garlan et al. 94a]
Garlan, D., M. Shaw, and J. Galmes.
Experience with a course on architectures for software systems, part II: educational materials.
Technical Report CMU-CS-94-178, Computer Science Department, Carnegie Mellon University,
December, 1994.
- [Garlan et al. 94b]
Garlan, D., R. Allen, and J. Ockerbloom.
Exploiting style in architectural design environments.
In Proceedings of Second ACM SIGSOFT Symposium on the Foundations of Software Engineering. ACM, December, 1994.
- [Garlan et al. 95]
Garlan, D., R. Allen, and J. Ockerbloom.
Architectural mismatch, or, why it's hard to build systems out of existing parts.
In Proceedings of the 17th International Conference on Software Engineering. ICSE, April, 1995.
- [Haines et al. 94]
Haines, N., D. Kindred, J.G. Morrisett, S.M. Nettles, and J.M. Wing.
Composing first-class transactions.
ACM Transactions on Programming Languages and Systems, Short Communications, November, 1994.

[Heintze et al. 96]

N. Heintze, N., D. Tygar, J. Wing, H.C. Wong.
Model Checking Electronic Commerce Protocols.
In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, pages 147-164.
November, 1996.
Oakland, CA.

[Horn 93]

Horn, B.
Constrained objects.
PhD thesis, Computer Science Department, Carnegie Mellon University, November, 1993.
Appears as technical report CMU-CS-93-154.

[Jackson 94a]

Jackson, D.
Abstract model checking of infinite specifications.
In *Proceedings of Conference on Industrial Benefit of Formal Methods Europe*. October, 1994.

[Jackson 94b]

Jackson, D.
Exploiting Symmetry in the Model Checking of Relational Specifications.
Technical Report CMU-CS-94-219, Computer Science Department, Carnegie Mellon University,
December, 1994.

[Jackson 94c]

Jackson, D.
Aspect: detecting bugs with abstract dependences.
In *Transactions on Software Engineering and Methodology*. ACM, 1994.

[Jackson 95a]

Jackson, D.
Structuring Z Specifications with Views.
*ACM Transactions on Software Engineering and Methodology*4(4), October, 1995.
Also appeared as technical report CMU-CS-94-126.

[Jackson 95b]

Jackson, D. and Damon, C.A.
Semi-executable Specifications.
Technical Report CMU-CS-95-216, Carnegie Mellon University,
November, 1995.

[Jackson 96a]

Jackson, D.
Nitpick: A checkable specification language.
In *Proceedings of the Workshop on Formal Methods in Software Practice*. January, 1996.

[Jackson 96b]

Jackson, D., Jha, S., and Damon, C.A.
Faster Checking of Software Specifications by Eliminating Isomorphs.
In *Proceedings of the Conference on the Principles of Programming Languages*. January, 1996.

[Jackson 96c]

Jackson, D. and Damon, C.A.

Elements of Style: Analyzing a Software Design Feature with a Counterexample Detector.
In *Proceedings of the International Symposium on Software Testing and Analysis*. January, 1996.

[Jackson 96d]

Jackson, D.

Automatic Analysis of Architectural Style.
July, 1996.
Submitted.

[Jackson 97]

Jackson, D., Jha, S.

Isomorph-free Model Enumeration: A New Method for Checking Relational Specifications.
January, 1997.
Submitted.

[Jackson and Jackson 96]

Jackson, D. and Jackson, M.

Problem Decomposition for Reuse.
Software Engineering Journal 11(1):11-30, January, 1996.
Also available as technical report CMU-CS-95-108, January, 1995.

[Jackson and Ladd 94]

Jackson, D. and D.A. Ladd.

Semantic diff: A tool for summarizing the effects of modifications.
In *Proceedings of the International Conference on Software Maintenance*. September, 1994.

[Jackson and Rollins 94a]

Jackson, D. and E.J. Rollins.

Abstract program dependences for reverse engineering.
Technical Report CMU-CS-94-169, Computer Science Department, Carnegie Mellon University,
July, 1994.

[Jackson and Rollins 94b]

Jackson, D. and E.J. Rollins.

A new model of program dependences for reverse engineering.
In *Second ACM Symposium on Foundations of Software Engineering*. ACM, December, 1994.

[Jackson and Rollins 94c]

Jackson, D. and E.J. Rollins.

Abstraction mechanisms for pictorial slicing.
In *Proceedings of the Workshop on Program Comprehension*. November, 1994.

[Jackson and Wing 96]

Jackson, D., Wing, J.M.

Lightweight Formal Methods.
In *IEEE Computer*. IEEE, April, 1996.

- [Jackson, Damon, and Jha 96]
Daniel Jackson, D., C. Damon, and S. Jha.
Faster Checking of Software Specifications.
In *Proceedings of the ACM Conference on Principles of Programming Languages*, pages 79-90. ACM, January, 1996.
St. Petersburg Beach, FL.
- [Jackson, Jha, and Damon 97]
Jackson, D., S. Jha, and C. Damon.
Isomorph-free Model Enumeration: A New Method for Checking Relational Specifications.
ACM Transactions on Programming Languages and Systems, 1997.
Accepted for publication, pending minor revisions.
- [Kindred 96]
Kindred, D., Wing, J.M.
Fast, Automatic Checking of Security Protocols.
In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, pages 41-52.
November, 1996.
Also available as Carnegie Mellon University technical report CMU-CS-96-173, Sep 1996.
- [Kindred and Wing 96]
Kindred, D. and J. Wing.
Fast, Automatic Checking of Security Protocols.
In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, pages 41-52. November, 1996.
Oakland, CA.
- [Knabe 95]
Knabe, F.C.
Language Support for Mobile Agents.
Technical Report CMU-CS-95-223, Carnegie Mellon University, 1995.
- [Leavens and Wing 96]
Leavens, G.T., Wing, J.M.
Protection from the Underspecified.
Technical Report CMU-CS-96-129, Computer Science Department, Carnegie Mellon University, April, 1996.
Also available as Iowa State University Department of Computer Science technical report TR96-04.
- [Leavens and Wing 97]
Leavens, G.T. and J.M. Wing.
Protective Interface Specifications.
In *Proceedings of TAPSOFT'97 - Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE, Lecture Notes in Computer Science 1214*, pages 520-534. ACM SIGSOFT, April, 1997.
To appear. Also available as Technical Report CMU-CS-96-129R.

[Liskov and Wing 93]

Liskov, B.H. and J.M. Wing.
Specifications and their use in defining subtypes.
In *Proceedings of OOPSLA '93*. OOPSLA, September, 1993.

[Liskov and Wing 94a]

Liskov, B.H. and J.M. Wing.
A behavioral notion of subtyping.
ACM Transactions on Programming Languages and Systems, November, 1994.

[Liskov and Wing 94b]

Liskov, B.H. and J.M. Wing.
Corrigenda to ECOOP '93 Paper.
*SIGPLAN Notice*29(4), 1994.

[Melton 97]

Melton, R., Garlan, D.
Architectural Unification.
January, 1997.
Submitted.

[Minamide et al. 96]

Minamide, Y., G. Morrisett, and R. Harper.
Polymorphic Closure Conversion.
In *Proceedings of the 1996 Conference on Principles of Programming Languages*. 1996.
An extended version of this paper will appear as a Carnegie Mellon Computer Science Department Technical Report.

[Monroe 96a]

Monroe, R.T. and Garlan, D.
Style-based Reuse for Software Architectures.
In *Proceedings of the Fourth International Conference on Software Reuse*. April, 1996.

[Monroe 96b]

Monroe, R.T.
Capturing Design Expertise in Customized Software Architecture Design Environments.
In *Proceedings of the Second International Software Architecture Workshop*. Second International Software Architecture Workshop, ACM Press, San Francisco, CA, October, 1996.

[Monroe 97]

Monroe, R.T., Kompanek, A., Melton, R., Garlan, D.
Architectural Styles, Design Patterns, and Objects.
In *IEEE Software*. IEEE, January, 1997.

[Monroe et al. 97]

Monroe, R., A. Kompanek, R. Melton, and D. Garlan.
Architectural Styles, Design Patterns, and Objects.
IEEE Software, January, 1997.

[Morrisett et al. 95]

Morrisett, G., M. Felleisen, and R. Harper.

Abstract Models of Memory Management.

In *Proceedings of the SIGPLAN-SIGARCH-WG2.8 Conference on Functional Programming and Computer Architecture*. SIGPLAN, June, 1995.

[Mummert et al. 94]

Mummert, L., J.M. Wing, and M. Satyanarayanan.

Using belief to reason about cache coherence.

In *Proceedings of the Symposium on Principles of Distributed Computing*. August, 1994.

Also appears as technical report, CMU-CS-94-151.

[Nettles 95]

Nettles, S.M.

Safe and Efficient Persistent Heaps.

PhD thesis, Computer Science Department, Carnegie Mellon University, December, 1995.

Available as technical report CMU-CS-95-225.

[Notkin et al. 93]

Notkin, D., D. Garlan, W.G. Griswold, and K. Sullivan.

Adding implicit invocation to languages: three approaches.

In *Proceedings of the JSSST International Symposium on Object Technologies for Advanced Software*. November, 1993.

[O'Callahan 95]

O'Callahan, R. and Jackson, D.

Detecting Shared Representations Using Type Inference.

Technical Report CMU-CS-95-202, Carnegie Mellon University, September, 1995.

[O'Callahan 97]

O'Callahan, R., Jackson, D.

Lackwit: A Program Understanding Tool Based on Type Inference.

In *Proceedings of the Eighteenth International Conference on Software Engineering*. 1997

ICSE, May, 1997.

To appear.

[O'Callahan and Jackson 97]

O'Callahan, R. and D. Jackson.

Lackwit: A Program Understanding Tool Based on Type Inference.

In *Proceedings of the International Conference on Software Engineering*. May, 1997.

Boston.

[Ockerbloom 96]

Ockerbloom, J.

Comprehending the Web: Scalable Object Oriented Techniques for Information Integration in Heterogeneous Information Systems.

1996.

[Rivera and Danylyszyn 95]

Rivera, J.G. and Danylyszyn, A.A.

Formalizing the Uni-processor Simplex Architecture.

Technical Report CMU-CS-95-224, Carnegie Mellon University, 1995.

[Shaw 94a]

Shaw, M.

Beyond objects: a software design paradigm based on process control.

Technical Report CMU-CS-94-154, Computer Science Department, Carnegie Mellon University, 1994.

Also appears as Software Engineering Institute Technical Report CMU/SE-94-TR-15.

[Shaw 94b]

Shaw, M.

Patterns for software architectures.

In Proceedings of First Annual Conference on the Pattern Languages of Programming.

August, 1994.

Also appears as a chapter in *Pattern Languages of Program Design*, J. Coplein and D. Schmidt (eds.), Addison-Wesley, 1995. pp. 453-462.

[Shaw 95a]

Shaw, M.

Coping with heterogeneity in software architecture.

1995.

Unpublished position paper for Dagstuhl Workshop on Software Architecture, February 1995.

[Shaw 95b]

Shaw, M.

Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging.

In Proceedings of the Symposium on Software Reuse, '95. 1995.

[Shaw 95c]

Shaw, M.

Some Patterns for Software Architecture.

In The Second Annual Conference on Pattern Languages of Programming. September, 1995.

[Shaw 95d]

Shaw, M. and Garlan, D.

Formulations and Formalisms in Software Architecture.

In Jan van Leeuwen (editor), Lecture Notes in Computer Science: Volume 1000. Springer-Verlag, 1995.

[Shaw 95e]

Shaw, M.

Making Choices: A Comparison of Styles for Software Architecture.

IEEE Software, Special Issue on Software Architecture 12(6):27-41, November, 1995.

[Shaw 96]

Shaw, M.

Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does.

In *Proceedings of the 8th International Workshop on Software Specification and Design*.
March, 1996.

[Shaw and Clements 97]

Shaw, M. and P. Clements.

A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems.

In *Proceedings of COMPSAC '97*. 1997.A preliminary version, "Toward Boxology: Preliminary Classification of Architectural Styles," also appears in *Proceedings of the Second International Software Architecture Workshop*, Oct, 1996.

[Shaw and Garlan 94]

Shaw, M. and D. Garlan.

*Characteristics of higher-level languages for software architecture.*Technical Report CMU-CS-94-210, Computer Science Department, Carnegie Mellon University,
December, 1994.

[Shaw and Garlan 96]

Shaw, M., Garlan, D.

Software Architecture: Perspectives on an Emerging Discipline.

Prentice Hall Publishing, Copyright 1996, 242 pp., Paper (0-13-182957-2), 1996.

[Shaw et al 96]

Shaw, M., Clements, P.

How Should Patterns Influence Architecture Description Languages?

July, 1996.

Response to a call for discussion among the DARPA EDCS community.

[Shaw et al. 95a]

Shaw, M., D. Garlan, R. Allen, D. Klein, J. Ockerbloom, C. Scott, and M. Schumacher.
Candidate Model Problems in Software Architecture.
1995.

This work, previously available on paper, now exists as a weblet:

<http://www.cs.cmu.edu/~ModProb/> .

[Shaw et al. 95b]

Shaw, M., R. DeLine, D.V. Klein, T.L. Ross, D. Young, and G. Zelesnik.

Abstractions for software architecture and tools to support them.

In *IEEE Transactions on Software Engineering*. IEEE, April, 1995.

[Shaw et al. 96]

Shaw, M., DeLine, R., and Zelesnik, G.

Abstractions and Implementations for Architectural Connections.

In *Proceedings of the Third International Conference on Configurable Distributed Systems*.
May, 1996.

[Wing 94]

Wing, J.M.

Decomposing and recomposing transactional concepts.

In Proceedings of the Workshop on Object-based Distributed Programming. Lecture Notes in Computer Science 791, 1994.

[Wing 95a]

Wing, J.M.

Hints for Writing Specifications.

In Proceedings of the Z Users' Meeting '95. September, 1995.

[Wing 95b]

Wing, J.M.

Teaching Mathematics to Software Engineers.

In Proceedings of the Fourth International Conference on Algebraic Methodology and Software Technology. AMAST, July, 1995.

Also available as Technical Report CMU-CS-95-118R, May 1995.

[Wing 96]

Wing, J.M.

*Educational Issues of Formal Methods.**In Hinchey, M., Dean, N.,**Academic Press, London, 1996, pages 57-77, Chapter 5, Hints to Specifiers.*

[Wing 97]

Wing, J.

Subtyping for Distributed Object Stores.

In Proceedings of the Second IFIP International Workshop on Formal Methods for Open Object-based Distributed Systems (FMOODS). IFIP, July, 1997.

Extended abstract of invited talk.

[Wing and Steere 95]

Wing, J.M. and D.C. Steere.

Specifying Weak Sets.

In Proceedings of the 15th International Conference on Distributed Computing Systems.

ICDCS, June, 1995.

Also available as Technical Report CMU-CS-94-194. October 1994.

[Wing and Vaziri-Farahani 95]

Wing, J.M. and M. Vaziri-Farahani.

Model Checking Software Systems: A Case Study.

In Proceedings of SIGSOFT Foundations of Software Engineering. SIGSOFT, March, 1995.

Also available as Technical Report CMU-CS-95-128. March 1995.

[Wing and Vaziri-Farahani 96]

Wing, J.M. and Vaziri-Farahani, M.

A Case Study in Model Checking Software Systems.

In Science of Computer Programming. September, 1996.

Also available as Carnegie Mellon University technical report CMU-CS-96-124.

[Wing and Vaziri-Farahani 97]

Wing, J. and M. Vaziri-Farahani.

A Case Study in Model Checking Software Systems.

Science of Computer Programming 28:273-299, 1997.

[Zaremski 96]

Zaremski, A.

Signature and Specification Matching.

Technical Report CMU-CS-96-103, Carnegie Mellon University, 1996.

[Zaremski and Wing 94]

Zaremski, A.M. and J.M. Wing.

Signature matching, a tool for software libraries.

ACM Transactions on Software Engineering and Methodology, 1994.

[Zaremski and Wing 95]

Zaremski, A.M. and J.M. Wing.

Specification Matching of Software Components.

In *Proceedings of 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*. SIGSOFT, October, 1995.

Also available as Technical Report CMU-CS-95-127. March 1995.

[Zaremski and Wing 96]

Zaremski, A.M., Wing, J.M.

Specification Matching of Software Components.

In *ACM Transactions on Software Engineering and Methodology*. ACM, June, 1996.

[Zaremski and Wing 97a]

Zaremski, A.M., Wing, J.M.

Specification Matching of Software Components.

In *ACM Transactions on Software Engineering and Methodology*. ACM SIGSOFT, January, 1997.

To appear.

[Zaremski and Wing 97b]

Zaremski, A. and J. Wing.

Specification Matching of Software Components.

ACM Transactions on Software Engineering and Methodology, 1997.

To appear, accepted May, 1997.

3. Integrated Software Architectures

Our research in Integrated Software Architectures (ISAs) has striven to unify problem solving, planning, communication, and machine learning to produce robust, knowledge-based systems that apply to a variety of externally-important domains. These domains might include autonomous robotic planning, logistics/transportation planning, distributed scheduling, planning under uncertainty, agent communication, etc. The research thrust has been primarily toward increasing the power and performance of these systems to meet real-world problems. Additionally, basic research into the component mechanisms continues, including investigations in nonlinear, hierarchical planning, large-scale matching, reactive decision making, and autonomous learning.

3.1 Cognitively-Oriented Task Simulation

We investigated the automated acquisition of task and domain knowledge via natural language descriptions as processed by NL-Soar.

Research under this contract focused on three distinct areas: the development of pervasive capabilities that can be reused by independently constructed task systems, enabling technology for large-scale Soar systems, and support of Soar for the external community. Progress along each direction is outlined below.

3.1.1 Pervasive capabilities for independent task systems

In [Lewis 93] we present the culmination of our previous work in producing a task-independent model of language comprehension based on functional and psycholinguistic constraints. NL-Soar is based on the Soar theory of cognitive architecture, which provides the underlying control structure, memory structures, and learning mechanism. The basic principles of NL-Soar are a result of applying these architectural mechanisms to the task of efficiently comprehending language in real time. We present a detailed computational model that provides in-depth accounts of structural ambiguity resolution, garden path effects, unproblematic ambiguities, parsing breakdown on difficult embeddings, unacceptable embeddings, immediacy of interpretation, and the time course of comprehension. The model explains a variety of both modular and interactive effects, and shows how learning can affect ambiguity resolution behavior. In addition to accounting for the qualitative phenomena surrounding parsing breakdown and garden path effects, NL-Soar explains a wide range of contrasts between garden paths and unproblematic ambiguities, and difficult and acceptable embeddings: The theory has been applied in detail to over 100 types of structures representing these contrasts, with a success rate of about 90%. The account of real time immediacy includes predictions about the time course of comprehension and a zero-parameter prediction about the average rate of skilled comprehension. Finally, the theory has been successfully applied to a suggestive range of cross-linguistic examples, including constructions from head-final languages such as Japanese.

In addition we have essentially completed the first integration of NL-Soar with an independently constructed task system, NTD-Soar, a model of the NASA test director [Nelson et al. 94a]. The combined NL/NTD system includes comprehension, our preliminary language generation model and our preliminary visual model. As our first integration it is not surprising that many unforeseen issues arose, changing the original NTD and NL systems to various degrees. From this

effort we have extracted a version of NL-Soar's comprehension capability to be plugged into the next application, an independently constructed tactical fighter pilot, Tacair-Soar. Although the generation framework will be preserved in the new integration as well (since the development of the capability itself is preliminary), we expect the details of that portion of the system to change significantly. It is unclear at this time whether we will attempt to integrate the visual model with Tacair-Soar or not (the decision depends upon whether the designers of the Tacair agent feel that modelling the visual limitations of the agent is desirable).

In [Nelson et al. 94b] we present a more detailed description of NTD-Soar than was referenced in our previous progress report. NTD-Soar was our first integration of NL-Soar with an independently constructed task system. NTD-Soar is a model of the perceptual, cognitive, and motor actions performed by the NASA Test Director as he prepares for a space shuttle launch. The model is based on a cognitive analysis of the NTD's task as well as a number of independently-designed, general cognitive capabilities, including language comprehension, language generation, and a preliminary model of visual attention. This paper presents a detailed description of the model and an assessment of its performance when compared to human data. Of particular importance is NTD-Soar's ability to display human-like, realtime performance. The comparison demonstrates that serial bottlenecks in symbolic models do not preclude complex behaviors that appear to happen in parallel, simply by opportunistically interleaving small elements of the different subtasks.

In [Rubinoff and Lehman 94] we focus on the new language generation capability in NL-Soar that is briefly mentioned in [Nelson et al. 94b]. In addition to discussing its deployment within the NTD system, we also discuss its deployment in TacAir-Soar, an independently-constructed, tactical fighter pilot built for ARPA's IFOR program. Like the NTD application, the TacAir agent requires language capabilities that work in a realtime environment. Responding in real time to changing situations requires a flexible way to shift control between language and task operations. NL-Soar's generation subsystem provides this flexibility by organizing generation as a sequence of incremental steps that can be interleaved with task actions as the situation requires. Despite the fact that the overall structure of the two agents is as different as possible (given their implementation within the Soar architecture), our preliminary generation capability was ported to the new application with little modification other than the addition of task-specific, linguistic knowledge.

Just as [Nelson et al. 94b, Rubinoff and Lehman 94] demonstrate that serial symbolic systems can meet real-time language constraints, [Lehman 94] shows how to subsume the power of statistical methods in coping with the computational complexity of sense resolution. Sense resolution, or word-sense disambiguation, is one of the two major sources (the other being syntactic ambiguity) of exponential blowup in natural language comprehension systems. The statistical basis for sense resolution decisions is achieved by applying a process to a corpus of instances. In general, once the process has been applied to the corpus, the system contains both some residual representation of the instances and some explicit augmentation of that representation with information that was implicit in the corpus. For example, part of the residual representation of "He feels happy on Fridays" might be the (*word sense*) pair (*happy feel-as-emotion*), and part of the augmentation might be the probability of "happy" co-occurring with the sense of "feel" as an emotion. We show that for the simple, symbolic residual representation of (*word sense*) pairs, the existence of such a representation in and of itself captures much of the regularity inherent in the data. We also show how viewing this residual representation as a form of episodic memory

can enable symbolic, knowledge-rich systems like NL-Soar to take advantage of this source of regularity in performing sense resolution.

In [Pelton and Lehman 94] we begin exploring a set of general mechanisms for managing goal-directed behavior over extended durations. The notion of an intelligent agent performing a complex set of behaviors over an extended period is a useful abstraction for both the NTD and TacAir domains, as well as being a target description for much of current, practical AI. Such an agent will, like people, have hundreds of goals at different levels of specificity that it pursues simultaneously. Clearly, it is computationally undesirable to have to track the progress of each goal at every moment, especially if the agent is interacting with an externally-dictated, realtime environment. Instead, we would expect that an agent somehow attend to only those goals that can make progress in its current environment or that are in conflict with those that can make progress. The basis of our exploration is the conjecture that the same mechanisms that are necessary for dealing with uncertainty in short-term behavior can be co-opted into providing the appropriate behavior over long intervals as well. This paper explains those mechanisms in the context of a simple, short-duration task. We begin by showing how the macro operator method for achieving a goal requiring multiple actions breaks down when formulating agent models that interact with an uncertain external world. A macro operator encapsulates a plan to reach an objective. Occasionally the objective will be found to be unachievable, requiring the macro operator and its plan to be rejected. Letting the macro operator interact with the external world does not, by itself, change this situation, but the fact that the results of the interaction are uncertain, and the agent's knowledge incomplete, does. The key idea is that the agent can't positively determine if progress towards the objective is being made in the external world, and thus it will err in rejecting a macro operator that would succeed. We show that there are a number of methods by which the agent can recover from such an operator rejection and continue toward the operator's objective. If we make operator rejection and recovery into a common mechanism, then the operators and the plans they represent will be split by the interaction into a sequence of smaller operators, each doing a portion of the work toward the objective of the larger operator. The application of these ideas to long duration objectives is clear when we observe that the key is managing the notion of progress in a way that allows for smooth continuation of a rejected or suspended goal.

The majority of our work on integrating general capabilities in independently constructed agents has centered on providing natural language comprehension and generation to agents in complex domains. We previously discussed our initial progress in integrating with a second agent, TacAir-Soar, an IFOR (Intelligent FORces) agent being deployed with the battlefield simulation domain. It had been our hope that the version of NL-Soar produced during the first integration with NTD-Soar (which modeled the NASA Test Director) would be nearly plug-compatible with TacAir-Soar. Because there are significant differences between the way the two agents are realized within the Soar architecture, this was not the case. Indeed, in many important ways, the two systems are at opposite ends of the structural spectrum that the Soar architecture permits [Soar 94]. In addition, the implementation of the Soar architecture itself has undergone a major revision, making our prior code incompatible with the current instantiation of the TacAir agent. The architecture now supports a conceptual model referred to as the NNPSM, as outlined in [Newell et al. 91].

The TacAir agent has been converted to run under NNPSM, and we have spent much of the current funding period converting NL-Soar as well. Although our progress has been slowed, our

current status is promising with respect to the contract goals. The NNPSM conversion is almost complete, and we have created a new version of NL-Soar that is, in theory, compatible with TacAir Soar *and*, retroactively, with NTD-Soar. What this means for our goal of plug-compatibility is that although we made an unfortunate choice in ordering our integrations (had we done TacAir first, the resulting NL-Soar would have fit into NTD-Soar with no problems), we have now created a version of the natural language capability that we believe will integrate smoothly with almost any Soar system (based on the maximal dissimilarity between NTD and TacAir on critical dimensions).

We would like to use this report as the vehicle to address two other issues relevant to independent capabilities as outlined in the contract. First, we had originally intended to pursue visual attention as our third capability, and have, in prior contract periods, integrated a model of visual attention into NTD-Soar. We have not found another independently-constructed Soar system whose designers believe they currently need a visual attention mechanism. On the other hand, the TacAir designers and a number of others who have been interested in our work with plug-compatible NL have urged us to explore speech recognition as our third capability. Consequently, we have started down this path and are currently exploring extensions to NL-Soar that will allow it to use the Sphinx-II system (developed at Carnegie Mellon under ARPA's HLT program) as part of the speech recognition capability. In order to bootstrap off existing speech databases, our preliminary work with this capability will be in the ATIS domain (airline scheduling).

The other issue concerns the nature of the third integration promised in the contract. We will be continuing work on other battlefield simulation agents and feel that these integrations meet the terms of the deliverables in principle. However, because the structure of these agents will be similar to TacAir-Soar, it is unclear how much we will learn about plug-compatibility from them. Thus, we have in mind pursuing another integration as well. This integration will be with a Soar agent that performs dynamic route-planning on the basis of radio information. We will look at using simulated radio broadcasts from local traffic reports as a first approximation to the true input. The agent will use this information to update a simple graphical display of the route a driver is following from home to work. We feel this integration is potentially exciting for three reasons. First, it will be the second testbed for the version of NL-Soar extended to handle speech recognition. Second, we believe it has obvious military applications (updating situation maps via helicopter reconnaissance). Third, and most important, we anticipate exploring yet another kind of agent during this integration. Specifically, our current plan is to integrate with a Soar agent that uses external software to do the actual route-planning and re-planning. Competition for I/O resources surrounding such a use of an external "device" will both teach us more about plug-compatibility and open up possibilities for integrating with other ARPA-sponsored, non-Soar systems.

In [Lehman et al. 95] we describe our efforts that support the Soar/IFOR project's goal of providing intelligent forces for distributed interactive simulation environments [Laird et al. 95]. The paper describes our initial progress in integrating the revised NL-Soar capability into an Air-IFOR agent. In particular we demonstrate how NL-Soar's linear complexity, interruptibility, and language processing atomicity provide a language comprehension and generation processes that do not compromise agent reactivity.

As we reported previously, the initial integration of NL-Soar with an independently constructed

tactical air agent led to some redesign of the original system as it had been integrated with a model of the NASA Test Director (NTD-Soar). We conjectured that we were creating a new version of NL-Soar that would be backwards compatible with NTD-Soar. We partially verified this expectation by reintegrating the most current version of the NL-Soar code with the NTD code, which had remained otherwise unchanged since August 1994. Despite the rather significant changes to the comprehension code during the intervening time, the reintegration of NL comprehension proved relatively painless. Only one of the required changes has any theoretical significance — the need to add explicit pause markers that allow NL-Soar to notice the ends of utterances. Interestingly, this pause marking (already present in TacAir's use of NL-Soar) matches perfectly to the transcription conventions already in place for the NTD dialogs. Pauses were received by NL-Soar after every 100 ms (or longer) pause in the transcript; a 100 ms duration was likewise used by the original transcribers to identify end-of-utterance units. The reintegration of generation proved more difficult, however, in part because the structure of generation was more heterogeneous than the structure of comprehension and in part because generation itself has not kept up with some of the changes to comprehension. As a result of this experience and our continued work in the tactical air domain, we have spent significant effort during this funding period redesigning and partially reimplementing NL-Soar's generation capability along the more robust lines of the comprehension code. Completing this reimplementation is our top priority in the coming six months.

We added significantly to NL-Soar in the areas of discourse knowledge and semantic representation. We have also begun to integrate with the Sphinx-II speech recognition system, and collected data necessary for our third integration.

The initial implementation of the language generation capability led to code that was more heterogeneous and less robust than the structure of the more mature comprehension capability. As a result of our on-going experience in integrating NL with independently constructed agents in the tactical air domain, we have spent significant effort in this funding period in the reimplementation of the generation capability along the more robust lines of the comprehension code. This effort is now essentially complete and the resulting capabilities now share code, data structures and theoretical perspectives that make the system as a whole easier to debug, maintain, and extend. One particularly interesting result of the language generation's redesign and reimplementation is that it now uses the comprehension capability directly as part of its processing. What this means, in practical terms, is that some of the learning done by the system during language generation transfers to comprehension (and vice versa), resulting in reduced training time for the system overall.

In addition, the initial extensions to NL comprehension and generation at the level of discourse processing have been the object of considerable progress. In [Green and Lehman 96a] we present a new methodology for acquiring compiled discourse knowledge and demonstrate that the resulting discourse knowledge can be used both in dialogue generation and interpretation. As in the case of reuse of the comprehension capability by generation, this automatic transfer of learned discourse knowledge between discourse generation and discourse comprehension should be usable in improving the training time of the system significantly. In [Green and Lehman 96b] we explore the current state of NL-Soar, and its discourse processing in particular, with respect to the difficult problem of conversational implicature— understanding the implied meaning behind what is actually stated in an utterance.

Finally, we have continued to push for a general, efficient solution to the problem of semantic processing across a variety of domains. The semantic representation, which now reflects Jackendoff's lexical conceptual structure, has been brought in line with the working memory data structures used by syntactic processing, again with significant code-sharing resulting.

We continued our work in creating a general language capability for intelligent Soar agents. This work has included both a continuation of our effort in the tactical air domain (in conjunction with John Laird's group at the University of Michigan) and new work in the domain of simultaneous interpretation. Although the tactical air domain depends largely on dialogue and the interpretation domain on narrative, both are critically characterized by their realtime demands and their need for both comprehension and generation capability. The same version of NL-Soar (NL96.04) is used in both applications.

With respect to the tactical air domain, [Green and Lehman 96c] presents our approach to agent modeling for communication and compares it to an approach for other types of action. The comparison is particularly instructive because both approaches are implemented in the same problem-solving architecture, face similar application domain requirements, and address the same general problem of comprehension. We show how it is possible for discourse processing to have the benefits of viewing agents in terms of their beliefs and intentions without sacrificing real-time performance and reactivity.

In [Green and Lehman 96d] we demonstrate how the architecture's learning mechanism affords a new methodology for automatically compiling discourse knowledge during discourse planning. The resulting, efficient form of discourse knowledge can be used both in dialogue generation and interpretation. We also show how NL96.04 unifies two disparate trends in discourse processing by allowing for both discourse recipes and compiled knowledge to coexist and change form within a common framework.

In [Lonsdale 96a, Lonsdale 96b] we discuss NL-Soar as an architectural account of the cognitive process of simultaneous interpretation (SI). The activity of SI, involving the simultaneous vocal translation of speeches from one language to another, is a complex cognitive task which involves many aspects of general and specialized language use. The system provide the first computational environment in which to examine many of the theoretical claims about the processes, skills, and tasks that are involved in SI. It also serves as a modelling tool to allow SI researchers to investigate cognitive performance of the SI task at lower levels than have been examined to date. As a baseline system it provides the framework for further computational approaches modelling capabilities not unique to SI but shared by other language-related tasks. A discussion of the relevant cognitive and SI research literature mentions skill components, empirical studies, and process models to be used. The proposed work follows other modelling efforts, in particular those involving language-task integration. We sketch the space of possible architectural constraints that can be posited to describe SI performance and, likewise, show how the NL-Soar framework can be leveraged in this effort. The goal of this research is the first low-level computational instantiation of the SI process. It fills a need to address SI research in a data-driven, bottom-up fashion to offset the top-down, anecdotal and experiential approach commonly used in SI studies. Useful information on control processes, integration of task components, the time-course of interpretation, and low-level automaticity will be derivable from SI/Soar based studies.

In addition, we have seen significant advances in the system's linguistic coverage (especially in

generation). In keeping with our change of focus from vision to speech, we have also made strides in extending NL-Soar to include an application-independent speech recognition capability by interfacing the system to the Sphinx-II speech recognizer. The core interface issue reflects a basic incompatibility in the contributing technologies; NL-Soar is an incremental natural language system and Sphinx-II is not (it assumes it has the entire sentence before it begins processing). To compensate, we have created a structure called the Rescorable Phonological Buffer (RPB) that permits Sphinx to work nonincrementally, while making the input appear incremental to NL-Soar. This construct offers the advantage of allowing feedback from NL-Soar (i.e., from higher-level language processes) to Sphinx on a word-by-word basis. Because NL-Soar learns continuously, such feedback is compiled over time so that the interaction can be fairly efficient.

We also pursued the goal of an independently constructed natural language comprehension and generation capability usable by other Soar agent builders.

Our effort to create a general language capability has occurred in both the tactical air domain (with John Laird's group at the University of Michigan) and in the domain of simultaneous interpretation; a new effort has begun in accounting for phenomena in second language acquisition. Note that all three task domains use the same version of NL-Soar (NL97.01).

In [Smith and Lehman 97], which has been accepted to a symposium on aviation communication at Embry-Riddle Aeronautical University, we report on current progress in providing natural language to independently-created autonomous intelligent forces (IFORS) for distributed interactive simulation environments in the tactical air combat domain. A long-term requirement for such simulations is that the non-human participants be effectively indistinguishable from their human counterparts. The capability for communicating in natural language (NL) is critical to meeting that requirement. In a time-critical domain such as air-combat, agent reactivity is paramount. For example, an agent must be able to quickly switch from an NL task (e.g., comprehension of a BRASH report) to a non-NL task (e.g., missile evasion) without having to wait for the NL task to terminate. In other cases it may be the NL-task which is more critical. NL-Soar's linear complexity (linear in the size of the message), interruptibility (time-critical tasks cannot be shut-out by less critical ones) and atomicity (interrupted processes are left in a consistent and resumable state) support this required reactivity. After summarizing NL-Soar's design, we demonstrate the practical progress of the system and discuss by way of examples the type of missions implemented, the coverage of NL communication, the syntactic, semantic, and discourse structures used, and current performance data.

In [Green and Lehman 97], we generalize the work in discourse comprehension begun in the tactical air domain, and propose an integrated architecture for discourse — generation, interpretation, and recipe acquisition — and describe an implementation of this discourse architecture in a dialogue system. In this approach, discourse recipes are acquired by compilation during discourse planning. We show that these compiled recipes both speed up discourse generation and can be exploited during discourse comprehension to recognize the speaker's discourse intentions.

In [Lonsdale 97] we present an overview of our approach to the study of simultaneous interpretation (SI), discussing current work in adding theoretical mechanisms and knowledge on top of the NL-Soar theory to permit study of lower-level cognition in interpretation. We give a sketch of the basic architecture of the total system as used in modelling comprehension. We then explore integration of the comprehension component with other tasks, and outline a unified

framework for studying the overall interpretation task. We discuss the system's potential for investigating various aspects of interpretation performance. This information, along with remarks about the more recent integration of NL-Soar's generation component, were delivered by invitation at an International Workshop on Methodological Issues in Interpreting Research, funded by the Swiss National Science Foundation and Swiss National Academy of Sciences, in Ascona, Switzerland.

With respect to the new work in second language acquisition, [Van Dyke and Lehman 97] describes a computational model based on NL-Soar that displays one class of errorful behavior of foreign-language learners. The particular area of concern is the English article system, which has been well-documented as difficult for learners of English as a foreign language. The paper provides a processing account that pinpoints the source of these errors in the architecture of the production system, using NL-Soar's control structure and learning mechanism to explain how they arise and to characterize the conditions required for overcoming them.

3.1.2 Speedup learning and large-scale systems

In [Doorenbos and Veloso 93] we examine a set of techniques for overcoming the utility problem in systems that use speedup learning. Speedup learning involves the acquisition of new knowledge from experience in order to improve the future performance of a problem-solver. We can differentiate between three aspects of this learning process: (1) the construction of new knowledge; (2) its storage and retrieval in the knowledge base; and (3) its reuse later. The tradeoff between the benefits of blindly increasing the size of the knowledge base and the costs of retrieving (or matching) its contents often leads to the utility problem (i.e. the cost of using the new knowledge may outweigh the savings it is supposed to represent). To avoid this, the speedup learning community has paid much attention to construction and reuse but has often overlooked the storage and retrieval aspects. Our recent work has studied the use of organization and indexing techniques at storage time, together with efficient matching or locating methods at retrieval time, to reduce or avoid the utility problem. We discuss our results in three speedup learning systems: Soar, Prodigy/Analogy, and Prodigy/EBL.

In addition to this comparative work, we pushed ahead with our evaluation of match optimizations in three additional Soar systems, each of which has learned more than 100,000 chunks without linear average growth in match cost. Indeed, one of the systems has been pushed to half a million productions without slowdown. To understand the progress this represents, recall that in 1992 we were the first to publish findings on system behavior at 10,000 productions, and in 1993 we were again the first, this time in examining behavior in a single system of 100,000 productions. With multiple systems at 100K productions and one at 0.5M productions, our current work remains in the forefront of large systems research. Further, the demonstration that the Soar architecture and its underlying implementation technology can support such large systems positions us to take on tasks that other architectures and paradigms may well find intractable.

In systems that learn a large number of rules (productions), it is important to match the rules efficiently in order to avoid the machine-learning utility problem. So we need match algorithms that scale well with the number of productions in the system. In our previous progress report, we noted that we had achieved our goal of sub-linear growth in match cost for a half-million production system. In [Doorenbos 94] we explain the theory and algorithms that have made this possible. Specifically, we review the notion of right unlinking as a way to improve the scalability of

the Rete match algorithm, and introduce a symmetric optimization, *left unlinking*, demonstrating that it makes Rete scale well on an even larger class of systems. Unfortunately, when left and right unlinking are combined in the same system, they can interfere with each other. We demonstrate a particular method of combining them, prove it minimizes this interference, and analyze the worst-case remaining interference. Finally, we present empirical results showing that the interference is small in practice, and that the combination of left and right unlinking allows five of our seven testbed systems to learn over 100,000 rules without incurring a significant increase in match cost.

In previous reports we have tracked progress in this area, watching as Soar systems were pushed from 10,000 productions to 100,000 productions to 500,000 productions over the last few years, without experiencing linear growth in the cost of matching the ever-expanding production memory. [Doorenbos 95] analyzes the problem, presents the algorithms that make this possible, and demonstrates the empirical results (including the performance of one system that grows to 1 million productions without slowdown in the match). This report marks the end of the research phase of this work, and the beginning of the transition of our research results into the release version of the architecture.

New to the area of scaling up are some results that have come from our work in creating a set of general mechanisms for managing goal-directed behavior over extended durations. This work, mentioned in our previous report under the heading of independent capabilities, has progressed to where our agent, Laureli, lives in its simple simulation world for more than 200 virtual days. The resulting growth in production memory has uncovered new areas of slowdown in the architecture that are *not* in the basic match procedure. In particular, Laureli is currently “reminded” of every goal in the past that is sufficiently similar to the one it is pursuing. Although the system never pursues these completed goals, the calculations that determine that they need not be pursued grow linearly in the number of days the system lives. Some of these calculations have been susceptible to straightforward improvements in the underlying algorithms (which will also be available in the next release). Others indicate areas for future research on learning so that Laureli’s memory for past events is partitioned in a more cost-effective way.

Work on scaling up shifted focus from Very Large Learning Systems (VLLSs) that solve many related problems to systems that reflect growth due to agent behavior over extended periods of time. Agents that exist over extended periods must be able to handle a large number of goals, opportunistically acting on them while maintaining a focus of purpose. They must also be able to operate over long lifetimes without significant degradation in performance. We have developed a general, domain-independent method of managing an agent’s goals so that the agent can make progress on as many goals as possible, without spending undue resources considering goals on which it cannot make progress ([Pelton and Lehman 95]). The key part of this method is a way of specializing and generalizing the set of perceptual cues that are used to bring a goal into consideration by the agent. Within the current funding period, this work has progressed from an initial, rudimentary design and implementation, to the maturity required for a thesis proposal. The proposed thesis includes empirical evaluation of the solution in the context of an internationally available, on-line interactive stock trading application, FASTWeb. Such an application environment will allow us to measure performance along specific dimensions of scaling, e.g., performance as a function of the total number of goals, as a function of goal conflicts, number of plans, etc.

Current progress has also included a number of performance improvements to the RETE-based pattern matcher that is at the core of Soar. We expect these improvements to be available in the next version of Soar, discussed below.

In [Pelton and Lehman 96] we continue our work on systems that reflect growth due to agent behavior over extended periods of time. People manage their many tasks by ignoring those that don't pertain to their current environment. This paper reports work aimed at making a computer agent (Laureli) handle its tasks in a similar manner. At any point Laureli considers only those tasks on which she can make progress in her current environment and those that are threatened by other tasks or environment circumstances. Laureli can manage her tasks in this way, learning when she should be considering each of her tasks, and attempting to suspend a task whenever she notices she is no longer making progress. Once she knows that she will appropriately remember a task, she then suspends the task by no longer considering it until the task is remembered. An agent that works in this manner has to make decisions at suspension time about what constitutes an appropriate set of conditions for remembering the task. Laureli uses her planning knowledge to create the conditions and modifies that knowledge depending upon her expectations about the future world.

3.1.3 Soar support

Soar 6, which reimplemented the architecture in C and was completed in the last half of the previous contract, is now in general use by the Soar community and available by anonymous ftp to all interested researchers. An updated manual has been made available as well.

We completed the work for the release of version 6.2.5 of the architecture. This will be the last non-NNPSCM release. Version 6.3.0 will support both the NNPSCM conceptual model and a TCL/TK interface to the system. With these fundamental changes, there will also be a new manual released.

We believed that the changes in Soar 6.3.0, i.e., supporting the NNPSCM view and providing a TCL/TK interface, would help make Soar more usable. Usability has been a constant concern and we have a number of on-going efforts (with other sources of support) that are attempting to remedy the situation. With respect to this contract, we have provided some support along these lines to a student who is studying Soar programmers. In [Altmann et al. 95] he explores how a skilled programmer, working on a natural task, navigates a large information display with apparent ease. Although tangential to the main thrusts of this contract, we believe this work will contribute to our knowledge of usability.

Work continued on Soar version 7.0, which will support the NNPSCM conceptual model (as explained in our previous report, but mentioned as version 6.3.0) and a TCL interface, on a variety of platforms. This version already includes the performance improvements in the RETE-based matcher that resulted from scaling-up (VLLS) research under this contract. In addition to compatibility with TCL, we have been working on compatibility with Tk and the creation of a GUI for easier interaction with the architecture. The new release format makes it easier to build the system on the numerous platforms we support and includes the implementation of and documentation for a newly designed, more consistent and "friendly" command set.

In [Altmann 96] we report on the completion of a detailed study of an expert Soar programmer

focussing on how she makes use of hidden external information, first recalling that it exists and then finding it. The dissertation investigates the memory phenomena involved in recalling that external information exists. We present data representing a programmer navigating to hidden features in a real-world task environment. We then present a model that accounts for this navigation by encoding and using simple episodic memories for having seen a feature. The model inherits constraints from the underlying cognitive architecture, which specifies that learning is passive and pervasive and that it creates simple memories that depend on the feature itself being present as a cue. The nature of these memories requires the model to recall features to its "mind's eye" as cues in order to retrieve them. This retrieval process requires domain knowledge: familiarity with features in order to imagine them and an idea of conditions under which it may be useful to recall having seen them. Recalling that a hidden feature exists prompts the model to scroll to that feature. Thus the model's access to external information is a function of passively-encoded episodic memories, and retrieval of these memories using knowledge. As a claim applied to people this process appears to overlap with a recently-published theory of *long-term working memory*. This theory proposes that experts, for example in chess, use long-term memory to expand their working memory in their domain of expertise. We propose a ubiquitous, episodic, long-term working memory, in which people store information about features with little effort, and from which they retrieve this information when it is relevant. As a study of a Soar programmer working in Soar the protocols also provide data for understanding the features of the language and environment that facilitate or make difficult the programming task.

3.2 High-Performance Planning and Learning

The main characteristic of the Prodigy research project has been that of addressing planning as a *glass box* decision-making process to be integrated with learning. Our long-term goal has been to enable Prodigy to address increasingly-complex, real-world planning problems. Towards this goal, we have been progressively enhancing the planner and the learning algorithms to cope with the challenges presented by the real world. Our work during this contract period has occurred in the following areas:

- Learning control knowledge for plan quality;
- Learning planning operators by observation and practice;
- Interaction of planning and a dynamically-changing external world;
- Comparison of planning algorithms with external ARPA sites;
- Similarity metrics for case retrieval over geometric domains;
- Theorem proving by analogy integrated with planning;
- Making efficient planning technology robust;
- Coping with incomplete and potentially incorrect domain knowledge.

We later centered our research in the Prodigy project on robust planning and on developing machine learning methods to integrate with advanced planning algorithms. Specifically, we focused on the following modules and methods:

- *Plan quality:* We concluded the development of a learning algorithm to acquire planning control knowledge to produce plans of high quality. We achieved very interesting results in an initial set of experiments in a complex process planning domain.
- *Learning plan action models:* We pursued our work on automatically acquiring the model of the planning actions by observation of a planning expert. The learner builds planning operators by observing the actions of the expert, and then refines the operators learned through its own practice.
- *Learning to reduce search:* We developed techniques to learn primary effects of operators. We also pursued research along the direction of trying to automate the identification of simple problems to learn from.
- *Analogical reasoning:* We applied our previously developed techniques for planning and learning by analogical/case-based reasoning to a route planning domain using real maps of the city of Pittsburgh. This study led to the introduction of a new similarity metric that can consider geometric constraints efficiently.
- *Planning:* We continue to make the Prodigy algorithm accessible to the planning community. We introduced a formalization of the Prodigy planning algorithm. We developed, as an extension of Prodigy, a new robust and flexible planning algorithm that can combine the advantages of least-commitment planning and state-space search.
- *Planning with external events:* We are developing a planning methodology for domains with uncertainty in the form of external events that are not completely predictable.
- *Prodigy-UI:* We developed a tcl/tk-based graphical user interface that permits users with no knowledge of Lisp (Prodigy's implementation language) to interact easily

with the planner. It was demonstrated at the DARPA meeting in San Diego and at the SISTO Symposium, and was very well received.

The Prodigy-UI includes:

- The planner: Prodigy4.0
 - Several different planning search strategies
 - Interactive building of planning domains
 - A large number of planning domains and problems
 - The ability to have user-driven selection of planning alternatives
 - Several learning modules in ongoing release stages, including Prodigy/Analogy, and Quality
 - Examples of probabilistic planning, Weaver (described below).
- *Probabilistic planning:* To handle probabilistic and nondeterministic aspects in many domains, we developed Weaver, an extension to the Prodigy planner that can handle probabilistic outcomes of actions and external events. We investigated how to use Prodigy's learning capabilities in the probabilistic setting. We also investigated probabilistic representations of uncertainty in the soccer-playing domain. We have extended Prodigy to reason about uncertain outcomes of actions and to plan for the possibility of uncontrollable external events, and have worked on the use of analogical reasoning and inductive learning to improve the efficiency of planning in the face of uncertainty.
 - *Real-world domains:* To demonstrate Prodigy's efficacy in the real world, we are developing (1) a navigation domain that uses Pittsburgh's actual street layout, including additional data such as construction zones and traffic patterns, (2) a robotic domain that plans for the Xavier mobile robot, (3) a set of soccer-playing robots to study collaborative and adversarial planning and (4) domains based on the control of industrial processes in chemical and power engineering.
 - *Learning methods:* We developed several learning methods that help Prodigy plan better. These include learning operators by observation, learning to generate higher-quality plans, learning the primary effects of operators, and automatically changing the domain representation to aid planning, as well as hybrid learning strategies and work in analogy and automatic abstraction.
 - *Planning, learning, and search algorithms:* The efficiency of planning algorithms underlying Prodigy and other systems across the spectrum of planning problems is not yet well understood. We continued our work of analyzing the properties of these algorithms and making comparisons. We are investigating the use of novel search techniques in planning and navigation. We have considered, in particular, the problem of navigating an unknown environment. We have also extended the Prodigy search algorithm to ensure its completeness. Finally, we began work on a system for selecting from a library of available search algorithms the one that is most appropriate for a given problem.
 - *Interleaving planning and execution:* In many domains execution may have to proceed before planning is completed, and this condition has a fundamental impact on the planning algorithms used. We continued our studies of this effect.
 - *Collaborative, mixed-initiative, and adversarial planning:* Work in multiagent planning is being done in the soccer-playing domain. In addition we have begun to study

mixed-initiative planning, where the agents may be both human and machine, allowing close collaboration between the human planner and automatic planning system.

- We study the problems of coordinating multiple independent agents and using them effectively against their adversaries, in the setting of a soccer-playing domain. We use real mini-robots in our exploration, thus combining the coordination problem with the task of real-world navigation.

3.2.1 Plan repair mechanisms

Real application domains require effective planning even when planning operators and other domain knowledge may be incomplete or inaccurate. For instance, military logistics planning is based on reliable information (such as terrain-topography, capabilities of transport vehicles, and so on), as well as less-than-reliable information (long-range weather forecasts, road conditions, enemy capabilities, etc.). Finally, some information may become available only at plan execution time (for instance, enemy tactical deployment), not at advance planning time.

In order to start coping with such situations in automated planning, present state-of-the-art planners—such as Prodigy—require several extensions. One such extension under active investigation is dynamic plan repair. In essence this strategy generates a plan from best-available information but then closely monitors its execution. When the execution reaches an impasse or deviates from the expected, the plan is repaired from the current state to overcome the impasse. The full power of the planner is reinvoked, as localized repairs (tried first) may not prove sufficient and substantive replanning may be required. In addition to immediate plan repair, Prodigy extends its domain knowledge through experience, so as to be better prepared to plan the next time the same knowledge is required, but now in its new, corrected or extended form.

The plan repair process has been successfully applied in the larger context of a learning-from-observation framework to a large machining-floor, process-planning domain.

3.2.2 Learning plan-optimization rules

Research on automatically acquiring control rules for plan para-optimization is progressing successfully. Although AI planners have improved substantially in recent years with respect to expressivity, generality, and efficiency of the planning process itself, less attention was paid to the execution-efficiency of the plan. Present research aims at learning rules to synthesize plans whose execution-efficiency is maximized. These rules are acquired from experience and through apprenticeship to experts. Preliminary results indicate that the methods do indeed work at reducing wasted action, at selecting the most appropriate (e.g. least cost) path among multiple possibilities, and at producing shorter plans. Quantitative results should be available by next reporting period.

3.2.3 Dissemination of Prodigy 4.0 architecture

Prodigy 4.0 is the robust underlying substrate for all of the present investigations, combining linear and nonlinear planning, single-level and hierarchical planning, deliberative and anytime planning, with several learning mechanisms. The Prodigy 4.0 planning substrate was completed last contract period, and it is being distributed to other ARPA research sites at their request. We are also maintaining and enhancing the basic system and its documentation as required.

3.2.4 Incremental learning of control knowledge for nonlinear problem solving

We developed a learning method that combines a deductive and an inductive strategy to learn control knowledge efficiently. The approach consists of initially bounding the explanation to a predetermined set of problem-solving features. Since there is no proof that the set is sufficient to capture the correct and complete explanation for the decisions, the control rules acquired are then refined, if and when applied incorrectly to new examples. The method is especially significant as it applies directly to nonlinear planning where the search space of possible plans is complete and therefore vast. We implemented our approach in Hamlet as a new inductive learning system within the context of the Prodigy architecture. The control rules for individual decisions that Hamlet learns correspond to new learning opportunities offered by the nonlinear problem solver. These opportunities go beyond those offered by the linear problem solver, and involve, among other issues, completeness, quality of plans, and opportunistic decision making. Preliminary empirical experiments yielded promising results illustrating Hamlet's learning performance.

3.2.5 Learning control knowledge for plan quality

In [Perez and Carbonell 94] we discuss our work in learning how to produce plans of better quality from experience. Generating production-quality plans is an essential element in transforming planners from research tools into real-world applications. However most of the work to date on learning planning control knowledge has been aimed at improving the *planning efficiency*—that is, learning to make the planner run faster by searching the solution space more efficiently. This earlier work—much of which was done with Prodigy, while other work was done outside Carnegie Mellon—has been termed “speed-up learning”. In contrast the new work focuses on learning control knowledge to guide a planner towards better solutions (i.e., to improve the *plan quality* as the planner's problem solving experience increases). We motivate the use of quality-enhancing search control knowledge and its automated acquisition from problem solving experience. We introduce an implemented mechanism for learning such control knowledge and some of our preliminary results in a process planning domain (see also [Perez 94a]).

In [Perez 94b] we present our learning approach in the context of its impact in goal-driven learning. The purpose of a learner is to make changes to a performance system to allow it to perform similar tasks more effectively the next time. The meaning of “more effectively” can be only asserted in the context of the learner's goals. Three types of learning goals can be distinguished in the context of planning systems: domain goals, planning efficiency goals, and plan quality goals. Most work to date on learning for problem solving has focused on the first two types. We present a learning mechanism that focuses on the third type, namely learning control knowledge to improve plan quality, and give some preliminary results on the use of this method in a process planning domain.

In [Borrajo and Veloso 94] we advocate a learning method in which deductive and inductive strategies are combined to efficiently learn control knowledge. The approach consists of initially bounding the explanation to a predetermined set of problem solving features. Since there is no proof that the set is sufficient to capture the correct and complete explanation for the decisions, the control rules acquired are then refined if and when applied incorrectly to new examples. The method is especially significant as it applies directly to nonlinear problem solving, where the search space is complete. In this paper, we also introduce Hamlet, a system in which we imple-

ment this learning method within the context of the Prodigy architecture. Hamlet learns control rules for individual decisions corresponding to new learning opportunities offered by the non-linear problem solver; these opportunities go beyond those afforded by the linear solver. These involve, among other issues, completeness, quality of plans, and opportunistic decision making. Finally, we show empirical results illustrating Hamlet's learning performance.

3.2.6 Learning domain knowledge by observation and practice

The work described in [Wang 94a] addresses learning planning operators by observing expert agents and subsequent knowledge refinement in a learning-by-doing paradigm. The observations of the expert agent consist of:

- The sequence of actions being executed
- The state in which each action is executed
- The state resulting from the execution of each action.

Planning operators are learned from these observation sequences in an incremental fashion utilizing a conservative specific-to-general inductive generalization process. In order to refine the new operators to make them correct and complete, the system uses the new operators to solve practice problems, analyzing and learning from the execution traces of the resulting solutions or execution failures. We describe techniques for planning and plan repair with incorrect and incomplete domain knowledge, and for operator refinement through a process that integrates planning, execution, and plan repair. Our learning method is demonstrated in multiple domains (see also [Wang 94b]).

In [Wang and Veloso 94] we discuss the interaction between learning domain and control knowledge by observation and practice. Acquiring planning knowledge from experts is a rather difficult knowledge engineering task. This work provides a novel method for accumulating domain and control planning knowledge by learning from the observation of expert planning agents and from one's own practice. Acquiring control knowledge that effectively copes with the incremental changes in the domain knowledge is a challenging problem. Our approach to this problem is to accumulate complete problem solving episodes, or "macros", from observation. These macros can then be reused and refined at practice time.

3.2.7 Planning in a dynamically-changing external world

In [Blythe 94a] we describe a planning methodology for domains with uncertainty in the form of external events that are not completely predictable. The events are represented by enabling conditions and probabilities of occurrence. The planner is goal-directed and backward chaining, but the subgoals are suggested by analyzing the probability of success of the partial plan rather than being simply the open conditions of the operators in the plan. The partial plan is represented as a Bayesian belief net to compute its probability of success. Since calculating the probability of success of a plan can be quite expensive, we introduce two other techniques for computing it, one that uses Monte Carlo simulation to estimate it and one based on a Markov chain representation that uses knowledge about the dependencies between the predicates describing the domain (see also [Blythe 94b]).

3.2.8 Comparison of planning algorithms

In [Veloso and Blythe 94] we compare Prodigy's nonlinear state-space based planner with SNLP, a nonlinear plan-space based planner. Recently, several researchers have demonstrated domains where partially-ordered planners outperform totally-ordered planners. In this paper, we demonstrate that totally-ordered planners sometimes have an advantage. We describe a series of domains in which Prodigy4.0 consistently outperforms SNLP and introduce the concept of "linkability" to characterize the class of domains for which this happens. Linkability highlights the fact that partially-ordered planners commit to causal links in much the same way that totally-ordered planners commit to step ordering.

In [Stone and Veloso 94a] we present our work on the need for different, domain-independent search heuristics for planning, such as those used by Prodigy's planning algorithm. We support our belief that no single search heuristic performs more efficiently than others for all problems or in all domains. The paper presents three different, domain-independent search heuristics of increasing complexity. We run Prodigy4.0 with these heuristics in a series of artificial domains where, in fact, one of the heuristics performs more efficiently than the others. However, we introduce an additional simple domain where the apparently worst heuristic outperforms the other two. The results we obtained in our empirical experiments lead to the main conclusion of this paper: Planning algorithms need different search heuristics in different domains. We conclude by advocating the need to learn the correspondence between particular domain characteristics and specific search heuristics for planning efficiently in complex domains.

3.2.9 Similarity metrics for case retrieval geometric domains

In [Haigh and Shewchuk 94] we present our work on defining similarity metrics to support efficient retrieval of planning cases for route planning in real maps. Case-based reasoning is a problem solving method that uses stored solutions to problems to aid in solving similar new problems. One of the difficulties of case-based reasoning is identifying cases relevant to a problem. If the problem is defined on a geometric domain—for instance, planning a route using a city map—it becomes possible to take advantage of the geometry to simplify the task of finding appropriate cases. We propose a methodology for determining a set of cases that *collectively* forms a good basis for a new plan and may include *partial* cases, unlike most existing similarity metrics. This methodology is applicable in continuous-valued domains, where one cannot rely on the traditional method of simple role-substitution and matching.

Our approach transforms the problem of identifying relevant cases into a geometric problem with an exact solution. We construct two similar algorithms for solving the geometric problem. The first algorithm returns a correct solution but is prohibitively slow. The second algorithm, based on the use of a Delaunay triangulation as a heuristic to model the case library, is fast and returns an approximate solution that is within a constant factor of optimum. Both algorithms return a good set of cases for geometric planning. We have implemented the second algorithm within a real-world robotics path planning domain.

3.2.10 Theorem proving by analogy integrated with planning

In [Melis and Veloso 94] we present our work on how “analogy makes proofs feasible.” Many mathematical proofs are hard for humans to generate and even harder for automated theorem provers. Classical techniques of automated theorem proving involve the application of basic rules, of built-in special procedures, or of tactics. Melis introduced a new method for analogical reasoning in automated theorem proving. In this paper we show how Veloso’s derivational analogy replay method may be related and extended to encompass analogy-driven, proof-plan construction. The method is evaluated by showing the proof-plan generation of the Pumping Lemma for context-free languages derived by analogy with the proof plan of the Pumping Lemma for regular languages. This is an impressive evaluation test for the analogical reasoning method applied to automated theorem proving, since the automated proof of this Pumping Lemma is beyond the capabilities of any of the current automated theorem provers.

3.2.11 Enabling efficient planning technology

In [Fink and Veloso 94] we present a formal description of the planning algorithm used in the Prodigy4.0 system. The algorithm is based on an interesting combination of backward-chaining planning and simulation of plan execution. The backward-chainer selects goal-relevant operators, then Prodigy simulates their application to the current state of the world. The system can use different backward-chaining procedures, some of which are presented in the paper.

In [de Silva 94] we discuss “goal-clobbering” avoidance in nonlinear planners. A central issue in nonlinear planning is the ordering of operators so as to avoid undesirable interactions between their effects. Goal-Clobbering Avoidance (GCA) attempts to avoid some interactions in a partially-ordered plan by promoting or demoting a sequence of operators, rather than individual operators. Effectively, it simultaneously applies Chapman’s Modal Truth Criterion to all operators in the sequence, using precompiled information about the domain. GCA has yielded large savings in planning time in numerous domains on the nonlinear planner Prodigy.

Finally we have been working on a new version of the Prodigy planner. This incarnation is implemented in C, as opposed to Lisp, and has a modified search algorithm that makes use of dynamic backtracking. Dynamic backtracking combines back jumping with caching to allow reusing information that would otherwise be lost and recalculated. The C version uses domain- and problem-input files that are not compatible with the Lisp version, but work is being done on a conversion program to move Lisp domain/problem definitions to the C format, and vice versa.

In conclusion, the first half of 1994 has proven to be an extremely productive time for the Prodigy project by almost any metric: infusion of new ideas, publications, scaling-up, recognition in the field, external sites requesting the programs, successful experiments, and so on.

3.2.12 Plan quality

Process planning poses significant computational requirements due to the variety of alternative processes, their complexity, and their interactions. General-purpose planners are not generally considered a practical approach, and most current research focuses on special-purpose planning systems. Our research aims to provide expressive, general-purpose planners, together with learning algorithms that can improve their efficiency, the accuracy of their domain model, and the quality of their plans [Gil and Perez 94]. Process planning is one of the large-scale, complex

domains that we have implemented in Prodigy to demonstrate the feasibility of our approach. Our current model of process planning is still far from comprehensive and is limited in many ways, but it reflects many of the complexities involved in the task. [Perez 94a] describes how Prodigy learns control knowledge, acquires domain knowledge, and improves the quality of its plans for this application domain using general-purpose planning and learning algorithms.

3.2.13 Learning plan action models

Acquiring knowledge from experts for planning systems is a difficult task, but is essential for any applications of planning systems. [Wang and Carbonell 94] addresses the issue of automatic acquisition of planning operators. Planning systems learn operators by observing the solution traces of expert agents, and subsequently refining knowledge in a learning-by-doing paradigm. Our approach is domain-independent and assumes minimal requirements for a priori knowledge and expert involvement in order to reduce the burden on the knowledge engineer and domain experts. Planning operators are learned from these observation sequences in an incremental fashion utilizing a conservative specific-to-general inductive generalization process. In order to refine the acquired operators to make them correct and complete, the system uses these operators to solve practice problems, analyzing and learning from the execution traces of the resulting solutions or execution failures. We describe techniques for planning and plan repair with incorrect and incomplete domain knowledge, and for operator refinement through a process which integrates planning, execution, and plan repair. Our learning method is implemented on top of the Prodigy architecture and has been demonstrated in the extended-STRIPS domain and a subset of the process planning domain [Wang 94b].

3.2.14 Learning to reduce search

Using primary effects of operators in planning is an effective approach to reducing planning time and improving solution quality. In the past, the characterization of "good" primary effects has remained at an informal level. No method has been known to automatically choose primary effects of operators for a given domain specification. In [Fink and Yang 94a, Fink and Yang 94b] we formalize the use of primary effects in planning, present a criterion for selecting useful primary effects that guarantee the efficiency and completeness of planning, and prove the near-optimality of solutions found by planning with primary effects. Based on the formalization we describe an algorithm for selecting automatically the primary effects of planning operators. We show that the algorithm performs efficiently and produces primary effects that enable a planner to generate near-optimal solutions with high probability. We also demonstrate empirically the effectiveness of the learned primary effects in reducing planning time.

Learning from past experience allows a problem solver to increase its solvability horizon from simple to complex problems. For planners learning involves a training phase during which knowledge is extracted from simple problems. But how are these simple problems constructed? All current learning and problem solving systems require the user to provide the training set. However, it is rarely easy to identify problems that are both simple and useful for learning, especially in complex applications. In [Stone and Veloso 94b] we present our initial research towards the automated or semi-automated identification of these simple problems. From a difficult problem and a corresponding, partially-completed search episode, we extract auxiliary problems with which to train the learner. We motivate this overlooked issue, describe our approach, and illustrate it with examples.

3.2.15 Analogical reasoning

Numerous applications can benefit from the ability to find optimal or good routes from real maps automatically. Several previous efforts therefore have endeavored to create and use real maps in computer applications. However, for the purpose of route planning, maps cannot be seen as static and complete, as there are dynamic factors and missing information that affect the selection of good routes, factors such as time of the day, traffic, construction, one- versus multi-lane roads, residential areas, etc. In [Haigh et al. 94] we describe our method for planning routes and dynamically updating the information available in a map. We show how we plan routes by reusing past routing cases that collectively form a good basis for generating a new routing plan. We briefly present our similarity metric for retrieving a set of similar routes. The metric effectively takes into account the geometric and continuous-valued characteristics of a city map. We then present how the planner produces the route plan by analogy with the retrieved similar past routes. Finally, we show how a real traversal of the route is a learning opportunity to refine the domain information and produce better routes. We illustrate our algorithms on a detailed online map of the city of Pittsburgh containing over 18,000 intersections and 25,000 street segments.

Case-based reasoning is a problem-solving method that uses stored problem solutions to aid in solving similar new problems. One of the difficulties of case-based reasoning is identifying cases relevant to a problem. If the problem is defined on a geometric domain—for instance, planning a route using a city map—it becomes possible to take advantage of the geometry to simplify the task of finding appropriate cases. In [Haigh and Shewchuk 94] we propose a methodology for determining a set of cases that collectively form a good basis for a new plan and may include partial cases, unlike most existing similarity metrics. This methodology is applicable in continuous-valued domains, where one cannot rely on the traditional method of simple role-substitution and matching. The problem of identifying relevant cases is transformed into a geometric problem with an exact solution. We construct two similar algorithms for solving the geometric problem. The first algorithm returns a correct solution, but is prohibitively slow. The second algorithm, based on the use of a Delaunay triangulation as a heuristic to model the case library, is fast and returns an approximate solution that is within a constant factor of optimum. Both algorithms return a good set of cases for geometric planning. We have implemented the second algorithm within a real-world robotics path planning domain.

3.2.16 Planning

In [Fink and Veloso 94], we present a formal description of the planning algorithm used in the Prodigy4.0 system. The algorithm is based on an interesting combination of backward-chaining planning and simulation of plan execution. The backward-chainer selects goal-relevant operators, and then Prodigy simulates their application to the current state of the world. The system can use different backward-chaining procedures, some of which we show.

Existing planners differ along several dimensions, including the nature of the search space, the plan step ordering commitments, and the use of the goal and initial states while planning. In particular some planners use a least-commitment strategy, i.e., they delay ordering commitments and search the space of possible plans. These planners reason from the initial state and from a set of ordering constraints that are regressed from the goal. Other planners commit eagerly to step orderings in order to make use of a uniquely specified state while planning. These planners back-chain from the goal and use their internal state to simulate plan execution. There has been

increasing evidence that neither commitment strategy can efficiently handle the complete spectrum of complex, real-world planning problems. In [Velo and Stone 95] we introduce a new planning algorithm that can use both types of commitment strategies, thus enlarging the range of problems that it can solve efficiently. Our new algorithm, FLECS, uses a flexible commitment strategy: It can either delay ordering commitments or commit eagerly and simulate an execution sequence. FLECS can vary its commitment strategy across different problems and domains and also during a single planning problem. FLECS represents a novel contribution to planning in that it introduces explicitly the choice of the commitment strategy to be used.

3.2.17 Planning with external events

In [Blythe 94a], we describe a planning methodology for domains with uncertainty in the form of external events that are not completely predictable. The events are represented by enabling conditions and probabilities of occurrence. The planner is goal-directed and backward chaining, but the subgoals are suggested by analyzing the probability of success of the partial plan rather than being simply the open conditions of the operators in the plan. The partial plan is represented as a Bayesian belief net to compute its probability of success. Since calculating the probability of success of a plan can be very expensive, we introduce two other techniques for computing it: one that uses Monte Carlo simulation to estimate it, and one based on a Markov chain representation that uses knowledge about the dependencies between the predicates describing the domain.

Probabilistic planners that use probabilities to represent and reason about uncertainty in the planning domain typically have a larger search space than their classical counterparts. Therefore heuristics that can reduce their search effectively are even more important. The "footprint" principle leads to a family of heuristics for probabilistic planners produced by attempting to make subsequent refinements to a plan apply to disjoint sets of states [Blythe 94c, Blythe 95]. Heuristics derived by this principle are shown to be effective for two probabilistic planners, Buridan and Weaver, which are organized around quite different search techniques.

3.2.18 Real-world domains

Automated route planning consists of using real maps in computer applications to find good map routes automatically. Most online map representations do not include information that may be relevant for the purpose of generating good realistic routes, such as traffic patterns, construction, and one-way streets. Furthermore, the notion of a good route is dependent on a variety of factors, such as the day of the week, the time of day, and may also be user-dependent. These factors may be acquired incrementally with route planning and execution experience. One promising approach we're exploring is analogical reasoning. Analogical reasoning, as an instance of case-based reasoning, is a method of using past experience to improve problem solving performance in new, similar situations.

Our work applies analogical reasoning to route planning through the accumulation and reuse of previously traversed routes [Haigh et al. 96]. We show how to exploit the geometric characteristics of this domain in the storage, retrieval, and reuse main phases of the analogical reasoning process. We demonstrate how our route planning method retrieves and reuses multiple, past routing cases that collectively form a good basis for generating a new routing plan. To find a good set of routes, we have designed a similarity metric that takes into account the geometric and continuous-valued characteristics of a city map. Our paper presents the replay mechanism and

the method the planner uses to produce a route plan by analogy with past routes retrieved by the similarity metric. We discuss the strategy used to merge a set of cases and generate a new route. We use illustrative examples and show some empirical results from a detailed online map of the city of Pittsburgh containing over 18,000 intersections and 25,000 street segments.

In [Stone and Veloso 95], we explore two advantages of interleaving execution with planning. First, the overall planning and execution time can be reduced. Second, information from the environment can be incorporated into the planner's knowledge of the world. We extend the Prodigy planner to handle execution as prompted by the user and to incorporate information that results from this execution. Such information can either arise automatically or can be input by the user. Finally, we briefly discuss ways to help the user determine potentially useful or necessary points for execution during planning.

In [Haigh and Veloso 96a] we describe Rogue, an integrated planning and executing robotic agent. Rogue is designed to be a roving office gopher unit, doing tasks such as picking up and delivering mail and returning and picking up library books, in a setup where users can post tasks for the robot to do. We have been working towards the goal of building a completely autonomous agent which can learn from its experiences and improve upon its own behavior with time. This paper describes what we have achieved to-date:

1. A system that can generate and execute plans for multiple interacting goals which arrive asynchronously and whose task structure is not known a priori, interrupting and suspending tasks when necessary,
2. A system which can compensate for minor problems in its domain knowledge, monitoring execution to determine when actions did not achieve expected results and replanning to correct failures.

In the Rogue system, we have developed an architecture that integrates high-level planning with a low-level executing robotic agent. Rogue is designed as the office "gofer" task planner for the Xavier robot. User requests are interpreted as high-level planning goals, such as getting coffee, and picking up and delivering mail or faxes. Users post tasks asynchronously and Rogue controls the corresponding planning and execution continuous process. In [Haigh and Veloso 97a] we present the extensions to a nonlinear state-space planning algorithm to allow interactions with the robot executor. We focus on presenting how executable steps are identified based on the planning model and predicted execution performance; how interrupts from user requests are handled and incorporated into the system; how executable plans are merged; and how monitoring execution can add more perceptual knowledge to the planning and possible needed re-planning processes. The complete Rogue system will learn from its planning and execution experiences to improve, with time, upon its own behavior.

Reliable and efficient execution of widely varied robot tasks in a dynamic world requires complex planning. Predicting all the eventualities and hand-coding corresponding behaviors is infeasible. We therefore aim at developing automated, machine-learning algorithms that examine real execution traces and extract relevant information with the goal of improving planning and execution behavior. In [Haigh and Veloso 97b] we present the content of an execution trace for the Xavier robot. We introduce the data representation of the path planning and navigation modules, emphasizing their differences and showing how the execution trace maps into the topological map used by the path planner. We describe the learning issues we address to handle the uncertainty and scale of the execution traces. We identify learning data from expectation

failures, and describe how situation-dependent features can be attached to the arc costs in the map and used to predict and avoid failure. Although preliminary, this work proved critical in setting the basic requirements for effective learning in a real-world framework with extensive continuous and probabilistic data. It provides the foundations for our ongoing development of the complete learning algorithm and further experimental work.

[Souza and Veloso 96a] describes an AI-planning-based framework to support the activities of a human operator in a supervisory control system. The framework uses an AI planning and learning substrate architecture and is designed for integration within general, third-party, realtime software. Our goal is to build a testbed architecture for research in AI planning applications, such as electrical and industrial processes. AI planning techniques, as opposed to more traditional, rule-based systems, can be useful in the automation of the supervision of process systems, as they provide rich planning representations and algorithms. We developed an AI planning system for a boiler power plant domain by first developing a set of planning operators from an extended multilevel flow modeling of the plant. Our planner reasons about goals and subgoals and generates plans for different scenarios, including the sequence for starting-up the plant. We show our approach to acquiring the domain knowledge, well-known as a difficult enterprise for real-world applications. We demonstrated that our modeling approach is successful in mapping the supervisory system knowledge into a planning representation.

3.2.19 Prodigy-UI

The Prodigy user interface supports the process of both building and running a planning domain in Prodigy. It was designed to be highly modular, requiring no changes to the code of the Prodigy planner to run, and extensible, so that interfaces to other modules built on Prodigy could be integrated easily into the interface. In [Blythe et al. 96] we describe how these goals were achieved. We demonstrate building a domain and animating the planning process. We describe extensions to the user interface to support planning by analogical reasoning and probabilistic planning with Prodigy.

3.2.20 Learning methods

Learning by observation and practice

[Wang 95] describes an approach to learning planning operators automatically by observing expert solution traces and to further refine the operators through practice in a learning-by-doing paradigm. This technique uses the knowledge naturally observable when experts solve problems, without need of explicit instruction or interrogation. Our approach differs from knowledge acquisition tools in that it does not require a considerable amount of direct interaction with domain experts. It differs from other work on automatically learning operators in that it does not require initial approximate planning operators or strong background knowledge.

The inputs to our learning system are: the description language for the domain, experts' problem solving traces, and practice problems to allow learning-by-doing operator refinement. Given these inputs, our system automatically acquires the preconditions and effects (including conditional effects and preconditions) of the operators. This representation is exactly what is used by most operator-based planners such as STRIPS, TWEAK, Prodigy, SNLP, and UCPOP. We present empirical results to demonstrate the validity of our approach in the process planning

domain. These results show that the system learns operators in this domain well enough to solve problems as effectively as human-expert coded operators.

Learning search control knowledge to improve plan quality

Generating good, production-quality plans is an essential element in transforming planners from research tools into real-world applications, but one that has been frequently overlooked in research on machine learning for planning. Most work has aimed at improving the *efficiency of planning* ("speed-up learning") or at acquiring or refining the planner's action model. [Perez 95] focuses on learning search-control knowledge to improve the *quality of the plans* produced by the planner.

Knowledge about plan quality in a domain comes in two forms: (a) a post-facto quality metric that computes the quality (e.g. execution cost) of a plan, and (b) planning-time decision-control knowledge used to guide the planner towards high-quality plans. The first kind is not operational until *after* a plan is produced, but is exactly the kind typically available—in contrast to the far more complex operational decision-time knowledge. Learning operational quality control knowledge can be seen as *translating* the domain knowledge and quality metrics into runtime decision guidance. The full automation of this mapping based on planning experience is the ultimate objective of this work.

Given a domain theory, a domain-specific metric of plan quality, and problems that provide planning experience, the *Quality* architecture we developed automatically acquires operational control knowledge that effectively improves the quality of plans generated. *Quality* can (optionally) learn from human experts who suggest improvements to the plans at the operator (plan step) level. We have designed two distinct domain-independent learning mechanisms to acquire quality control knowledge efficiently. They differ in the language used to represent the learned knowledge, namely control rules and control knowledge trees, and in the kinds of quality metrics for which they are best suited.

The *Quality* learning mechanism is fully implemented on top of the Prodigy4.0 nonlinear planner. Empirical evaluation has shown that the learned knowledge produces near-optimal plans (reducing before-learning plan execution costs 8% to 96%). Although the learning mechanisms and learned knowledge representations have been developed for Prodigy4.0, the framework is general and addresses a problem that must be confronted by any planner that treats planning as a constructive decision-making process.

Integrating planning and learning

Planning is a complex reasoning task well suited to the study of improving performance and knowledge by learning, i.e. by accumulation and interpretation of planning experience. Prodigy is an architecture that integrates planning with multiple learning mechanisms. Learning occurs at the planner's decision points, and integration in Prodigy is achieved via mutually interpretable knowledge structures. [Veloso et al. 95] describes the Prodigy planner, briefly reports on several learning modules developed earlier in the project, and presents in more detail two recently explored methods to learn to generate plans of better quality. We introduce the techniques, illustrate them with comprehensive examples, and show preliminary empirical results. The article also includes a retrospective discussion of the characteristics of the overall Prodigy architecture and discusses their evolution within the goal of the project of building a large and robust integrated planning and learning system.

Design of representation-changing algorithms

The performance of all problem-solving systems depends crucially on problem representation. The same problem may be easy or difficult to solve depending on the way we describe it. Researchers have designed a variety of learning algorithms that deduce important information from the description of the problem domain and use the deduced information to improve the representation. Examples of these representation improvements include generating abstraction hierarchies, replacing operators with macros, and decomposing complex problems into sub-problems. There has, however, been little research on the common principles underlying representation-improving algorithms and the notion of useful representation changes has remained at an informal level.

In [Fink 95a, Fink 95b], we present preliminary results on a systematic approach to the design of algorithms for automatically improving representations. We identify the main desirable properties of such algorithms, present a framework for formally specifying these properties, and show how to implement a representation-improving algorithm based on the specification of its properties. We illustrate the use of this approach by developing novel algorithms that improve problem representations.

Planning with primary effects

The use of primary effects in planning is an effective approach to reducing search [Fink and Yang 95]. The underlying idea of this approach is to select certain "important" effects among those of each operator, and to use an operator only for achieving its important effects. In the past, there has been little analysis of planning with primary effects and few experimental results. We provide empirical and analytical results on the use of primary effects. First, we experimentally demonstrate that the use of primary effects may lead to an exponential reduction of the planning time. Second, we analytically explain the experimental results and identify the factors that influence the efficiency of planning with primary effects. Third, we describe an application of our analysis to predicting the performance of a planner for a given selection of primary effects.

Integrating planning and learning

In [Wang 96] we describe issues that arise when integrating a planner with a system that learns planning operators incrementally, and discuss our approaches to address these issues. During learning, domain knowledge can be incomplete and incorrect in different ways; therefore the planner must be able to use incomplete domain knowledge. This presents the following challenges for planning: How should the planner effectively generate plans using incomplete and incorrect domain knowledge? How should the planner repair plans upon execution failures? How should planning, learning, and execution be integrated? This paper describes how we address these challenges in the framework of an integrated system, called Observer, that learns planning operators *automatically* and *incrementally*. In Observer, operators are learned by observing expert agents and by practicing in a learning-by-doing paradigm. We present empirical results to demonstrate the validity of our approach in a process planning domain. These results show that practice, in using our algorithms for planning with incomplete information and plan repair, contributes significantly to the learning process.

In [Perez 96] we describe Quality, a domain-independent architecture that learns operational quality-improving search-control knowledge given a domain theory, a domain-specific metric of plan quality, and problems which provide planning experience. Quality can (optionally) interact

with a human expert in the planning application domain who suggests improvements to the plans at the operator (plan step) level. The framework includes two distinct domain-independent learning mechanisms which differ in the language used to represent the learned knowledge, namely control rules and control knowledge trees, and in the kinds of quality metrics for which they are best suited. Quality is fully implemented on top of the Prodigy4.0 nonlinear planner and its empirical evaluation has shown that the learned knowledge is able to substantially improve plan quality.

[Veloso and Aamodt 96] contains the papers presented at the First International Conference on Case-based Reasoning, ICCBR-95, held on October 23-26, 1995, in Sesimbra, Portugal. (Agnar Aamodt and Manuela Veloso served as the conference program chairpersons.) ICCBR-95 marks the start of a joint CBR conference series that will follow and extend the CBR workshops that have taken place in the United States since 1988 and Europe since 1993. The overall aim of ICCBR-95 was to advance the scientific and application-oriented state of the CBR field by bringing researchers and system builders together for presentation of results and discussions of problem issues. The papers fell into the following categories: case and knowledge representation; case retrieval; nearest neighbor methods; case adaptation and learning; cognitive modeling; integrated reasoning methods; and application-oriented methods.

This work is now available as a book chapter, [Veloso 96a].

General-purpose generative planners use domain-independent search heuristics to generate solutions for problems in a variety of domains. However, in some situations these heuristics force the planner to perform inefficiently or obtain solutions of poor quality. Learning from experience can help to identify the particular situations for which the domain-independent heuristics need to be overridden. Most of the past learning approaches are fully deductive and eagerly acquire correct control knowledge from a necessarily complete domain theory and a few examples to focus their scope. These learning strategies are hard to generalize in the case of nonlinear planning, where it is difficult to capture correct explanations of the interactions among goals, multiple planning operator choices, and situational data. In [Borrajo and Veloso 96] we present a lazy learning method that combines deductive and inductive strategies to learn control knowledge efficiently and incrementally with experience. We present Hamlet, a system that learns control knowledge to improve both search efficiency and the quality of the solutions generated by a nonlinear planner, namely Prodigy4.0. We have identified three lazy aspects of our approach, from which we believe Hamlet greatly benefits: lazy explanation of successes, incremental refinement of acquired knowledge, and lazy learning to override only the default behavior of the problem solver. We show empirical results that support the effectiveness of this overall lazy learning approach in terms of improving the efficiency of the problem solver and the quality of the solutions produced.

Using primary effects of operators provides an effective approach to improving the efficiency of planning. The characterization of "good" primary effects, however, has remained at an informal level and there have been no algorithms for selecting primary effects of operators. In [Fink and Yang 96] we formalize the use of primary effects in planning and present a criterion for selecting useful primary effects, which criterion guarantees efficiency and completeness. We analyze the efficiency of planning with primary effects and the quality of the resulting plans. We then describe a learning algorithm that automatically selects primary effects and demonstrate, both analytically and empirically, that the use of this algorithm significantly reduces planning time and does not compromise completeness.

3.2.21 Planning algorithms

Total-order planners perform backtracking search over all possible total-orderings, but their commitment to a total order gives them a well-defined current state that can be used to guide the search at later choice points. Partial-order planners handle operator orderings more efficiently, but lack a current state. In such planners, an open condition can be linked to many steps (both pre-existing and newly introduced). In earlier work, we showed how the lack of a current state to give guidance at these choice points causes *linkability* difficulties. [de Silva 95] explores how partial-order planners can maintain an approximation of the current state and use it to give guidance at linkability choice-points. An inexpensive domain-independent heuristic selectively delays linking commitments in a manner very similar to that in which a total-order planner handles goals that are already satisfied in the current state. The heuristic, implemented in the partial-order planner UCPOP, permits it to avoid linkability difficulties to some extent. In effect, the heuristic embodies some of the strengths of total-order planners without inheriting their weaknesses.

The Prodigy project is primarily concerned with the integration of planning and learning. Members of the Prodigy research group have developed many learning algorithms for improving planning efficiency and plan quality, and for automatically acquiring knowledge about the properties of planning domains. The details of the Prodigy planning algorithm, however, have not been described in the literature. In [Fink and Veloso 95] we present a formal description of the planning algorithm used in the current version of the Prodigy system. The algorithm is based on an interesting combination of backward-chaining planning with simulation of plan execution. The backward-chainer selects goal-relevant operators and then the planner simulates the application of these operators to the current state of the world. The system can use different backward-chaining algorithms, two of which are presented in the paper.

3.2.22 Planning, learning, and search algorithms

The Pigeonhole Principle (PHP) has been one of the most appealing methods of solving combinatorial optimization problems. Variations of the Pigeonhole Principle, sometimes called the "Hidden" Pigeonhole Principle (HPHP), are even more powerful and often produce the most elegant solutions to nontrivial problems. However, some OR approaches, such as Linear Programming Relaxation (LPR), are strong competitors to PHP and HPHP. The OR approaches can also be applied to combinatorial optimization problems to derive upper bounds. It has been an open question whether PHP or LPR establish tighter upper bounds and how efficiently when applied to the same problem. Challenged by this open question, we identify that the main reason for the lack of ability to compare the efficiency of PHP and LPR is the fact that different problem representations are required by the two methods.

In [Smirnov and Veloso 96] we introduce a method to change problem representation into an integer programming form, allowing an alternative way to solve combinatorial problems. We also introduce several combinatorial optimization problems and show how to perform representation changes to convert the original problems into the integer programming form. Using the new problem model, we redefine the Pigeonhole Principle as a method of solving integer programming problems, determine the difference between PHP and HPHP, prove that PHP has the same bounding power as LPR (thus answering the above "open question"), and demonstrate that HPHP and integer cuts are actually similar representation changes of the problem domains.

If a state space is not completely known in advance, then search algorithms have to explore it sufficiently to locate a goal state and a path leading to it, performing therefore what we call goal-directed exploration. Two paradigms of this process are pure exploration and heuristic-driven exploitation: The former approach explores the state space using only knowledge of the physically visited portion of the domain, whereas the latter approach relies totally on heuristic knowledge to guide the search towards goal states. Both approaches have disadvantages: The first one does not utilize available knowledge to cut down the search effort, and the second one relies too much on knowledge, even if such is misleading. We have developed a framework for goal-directed exploration called VECA [Smirnov et al. 96] that combines the advantages of both approaches by automatically switching from exploitation to exploration on parts of the state space where exploitation does not perform well. VECA provides better performance guarantees than previously studied heuristic-driven exploitation algorithms, and experimental evidence suggests that this guarantee does not deteriorate average-case performance.

In addition, the work on formalizing the Prodigy algorithm, reported earlier, has now been published as a book chapter [Fink and Veloso 96], thereby increasing its availability.

Recently, there has been an increasing interest in understanding the capabilities of different planners. While there has been some work done on comparing planning algorithms, relatively little effort has been spent on planning domain representation. In [Perez et al. 96], we describe our experience with converting hierarchical task network (HTN) domains into an operator-based planner representation. We enumerate several features of the HTN representation and their corresponding definitions in the operator-based one. This work is based on an empirical study using the SIPE and Prodigy domains. We address the general question of expressivity of each of these planning frameworks and present a general, domain-independent transformation from a domain representation of a formalized HTN planner into Prodigy. We identify the capabilities of Prodigy that make this transformation feasible.

GSAT is a local hill-climbing procedure for solving propositional satisfiability problems. In [Smirnov and Veloso 97] we restate it as a navigational search process performed on an N-dimensional cube by a fictitious agent with limited lookahead. Several variations of GSAT have been introduced and provide different levels of efficiency, hence raising the interesting question of understanding the essence of their differing performance. We show how we use our navigational approach to investigate this issue. We introduce new algorithms that focus on specific combinations of properties of efficient GSAT variants and help identify the relevance of the algorithm features to the efficiency of the GSAT search. We derive fast approximating procedures based on variable weights that can provide good switching points for a mixed search policy. Our conclusions are validated by empirical evidence obtained from the application of several GSAT variants to random 3SAT problem instances and to simple navigational problems.

The Prodigy system is based on bidirectional planning, which is a combination of goal-directed, backward chaining with simulation of plan execution. Experiments have demonstrated that it is an efficient technique; a fair match to other successful planning systems. The question of completeness of bidirectional planning, however, has remained unanswered. In [Fink and Blythe 97] we show that Prodigy is not complete and discuss the advantages and drawbacks of its incompleteness. We then develop a complete bidirectional planner and compare it experimentally with Prodigy. We demonstrate that the new planner is almost as efficient as Prodigy and can solve more problems.

Choosing an appropriate problem-solving method from available methods is a crucial skill for experts in many areas. In [Fink 97] we describe a technique for automatic selection among methods, based on statistical analysis of their past performances. We formalize the statistical problem involved in selecting an efficient problem-solving method, derive a solution to this problem, and describe a method-selection algorithm. The algorithm not only chooses among available methods, but also decides when to abandon the chosen method if it proves too slow. We give empirical results on the use of this technique in selecting among search engines in the Prodigy problem-solving system.

3.2.23 Probabilistic planning

Probabilistic back-chaining planners, which use probabilities to represent and reason about uncertainty in the planning domain, typically have a larger search space than their classical counterparts. Therefore, heuristics that can reduce their search effectively are even more important. The “footprint” principle leads to a family of heuristics for probabilistic planners produced by attempting to make subsequent refinements to a plan apply to disjoint sets of planning cases. In [Blythe 95] heuristics derived by this principle are shown to be effective for two probabilistic planners, Buridan and Weaver, which are organized around quite different search techniques. Probabilistic planners are needed that can use more compact representations of uncertainty than those that currently exist, and these planners will depend even more on the footprint principle and others like it.

An increasing number of planners can handle uncertainty in the domain or in action outcomes. However, less work has addressed building plans when the planner’s world can change, independently of the planning agent, in an uncertain manner. In [Blythe 96a] we model this change with external events that concisely represent some aspects of structure in the planner’s domain. This event model is given a formal semantics in terms of a Markov chain, but probabilistic computations from this chain would be intractable in real-world domains. We describe a technique, based on a reachability analysis of a graph built from the events, that allows abstractions of the Markov chain to be built to answer specific queries efficiently. We prove that the technique is correct. We have implemented a planner that uses this technique, and show an example from a large planning domain.

An important direction for AI planning systems is to handle dynamic domains—domains where changes take place that are not the direct result of the actions of the agent, and where the world describes a state trajectory in the absence of any actions from the planning agent. In many real-world planning problems, for example, change to the world takes place due to other agents, or due to forces of nature over time. The planning agent may have limited control over these external events, and limited knowledge about both events that *have* occurred and events that *may* occur in the future.

The aim of [Blythe 96b] is twofold: To give an account of an action model for dynamic domains that is used in an implemented planner, and to show how issues of representational power and computational efficiency jointly influenced the model. A model for dynamic domains for planners should have at least the following three features: First, the planner should be able to take advantage of possible events, incorporating them into a plan as appropriate. Second, the model should allow the planner to reason about a set of different possible futures, each being consistent with the agent’s knowledge of events. Third, the planner should be able to efficiently determine

any events that may effect candidate partial plans, and obtain information that may allow a modified plan to be made safe from the events.

Probabilistic and nondeterministic aspects complicate planning in many domains. Weaver is a hybrid planning algorithm we've developed that can create plans in domains that include uncertainty, modelled either as incomplete knowledge of the initial state of the world, of the effects of plan steps or of the possible external events. The plans are guaranteed to exceed some given threshold probability of success. Weaver creates a Bayesian network representation of a plan to evaluate it, in which links corresponding to sequences of events are computed with Markov models. As well as generating the probability of success, evaluation identifies a set of flaws in the candidate plan, which are then used by the planner to improve the plan. In [Blythe and Veloso 96] we describe a learning method that generates control knowledge compiled from this probabilistic evaluation of plans. The output of the learner is search control knowledge for the planning domain that helps the planner select alternatives that have previously lead to plans with high probability of success. The learned control knowledge is incrementally refined by combined deductive and inductive reasoning.

Inaccessible and nondeterministic environments are common in realworld problems. One of the difficulties in these environments is representing the knowledge about the unknown aspects of the state. In [Bowling et al. 96] we present a solution to this problem for one such domain, robotic soccer. We developed a predictive memory model that builds a probabilistic representation of the state based on past observations. By making the right assumptions, we can create an effective model that can store and update knowledge for even the inaccessible parts of the environment. We conducted experiments to compare the effectiveness of our approach with a simpler approach, which ignored the inaccessible parts of the environment. The experiments consisted of using the memory models in a "free ball" situation, where two players race for the ball to pass or kick it to one of their teammates or the goal. The results obtained demonstrate that this predictive approach does generate an effective memory model that outperforms a non-predictive model.

The work on heuristics for Probabilistic planners, reported earlier, has been published as a book chapter, [Blythe 96c].

Several recent planners can create conditionally branching plans to solve problems involving uncertainty. These planners represent an important step in broadening the applicability of AI planning techniques, but they typically must search a larger space than nonbranching planners, since they must produce valid plans for each branch considered. In the worst case this situation can produce an exponential increase in the complexity of planning. If conditional planners are to become usable in real-world domains, this complexity must be controlled by sharing planning effort among branches. Analogical plan reuse should play a fundamental role in this process. In [Blythe and Veloso 97] we describe a conditional probabilistic planner that uses analogical plan replay to derive the maximum benefit from previously solved branches of the plan. This approach provides valuable guidance for when and how to merge different branches of the plan, and exploits the high similarity between different branches in a conditional plan that have the same goal and, typically, a similar state. Analogical replay can be applied to a variety of conditional planners, complementing the plan sharing that they may perform naturally. We present experimental data in which analogical plan replay significantly reduces the complexity of conditional planning.

3.2.24 Interleaving planning and execution

[Haigh and Veloso 96b] describes Rogue, an integrated planning and executing robotic agent. Rogue is designed to be a roving office "gopher" unit, doing tasks such as picking up and delivering mail or returning and picking up library books, in a configuration where users can post tasks for the robot. We have been working towards a completely autonomous agent that can learn from its experience and improve upon its own behavior with time. The paper describes what we have achieved to-date: (1) a system that can generate and execute plans for multiple, interacting goals (that arrive asynchronously and whose task structure is not known beforehand), interrupting and suspending tasks when necessary, and (2) a system that can compensate for minor problems in its domain knowledge, monitoring execution to determine when actions did not achieve expected results and replanning to correct failures.

The work on user-guided interleaving of planning and execution, reported earlier, has now been published as a book chapter, [Stone and Veloso 96a].

3.2.25 Collaborative, mixed-initiative and adversarial planning

[Veloso 96b] introduces our work on mixed-initiative, rationale-supported planning. The work centers on the principled reuse and modification of past plans by exploiting their justification structure. The goal is to record as much as possible of the rationale underlying each planning decision in a mixed-initiative framework where human and machine planners interact. This rationale is then used to determine which past plans are relevant to a new situation, to focus the user's modification and replanning on different relevant steps when external circumstances dictate, and to ensure consistency in multi-user distributed scenarios. We build upon our previous work in Prodigy/Analogy, which incorporates algorithms to capture and reuse the rationale of an automated planner during its plan generation. To support a mixed-initiative environment, we have developed user interactive capabilities in the Prodigy planning and learning system. We are also working towards the integration of the rationale-supported plan reuse in Prodigy/Analogy with the plan retrieval and modification tools of ForMAT. Finally, we have focused on the user's input to the plan reuse process, in particular when conditional planning is needed.

Distributed Artificial Intelligence (DAI) is concerned with systems that consist of multiple, independent entities that interact in a domain, and has existed as a subfield of AI for less than two decades. Traditionally, DAI has focussed on the information management aspects of these systems. But in the past few years, a subfield of DAI focussing on behavior management, as opposed to information management, has emerged. This young subfield is called Multiagent Systems (MAS). In [Stone and Veloso 96b] we present a survey of MAS intended to serve as an introduction to the field and as an organizational framework. It contains guidelines for when and how MAS should be used to build complex systems. A series of increasingly complex general multiagent scenarios are presented. For each scenario the issues that arise are described along with a sampling of the techniques that exist to deal with them. The presented techniques are not exhaustive, but they highlight how multiagent systems can be and have been used to build complex systems. When options exist, the techniques presented are biased towards Machine Learning approaches. Additional opportunities for applying Machine Learning to MAS are highlighted and robotic soccer is presented as an appropriate testbed for MAS.

Soccer is a rich domain for the study of multiagent learning issues. Not only must the players

learn low-level skills, but they must also learn to work together and to adapt to the behaviors of different opponents. We are using a robotic soccer system to study these different types of multiagent learning: low-level skills, collaborative, and adversarial. In [Stone and Veloso 96c] we describe in detail our experimental framework. We present a learned, robust, low-level behavior that is necessitated by the multiagent nature of the domain, namely "shooting" a moving ball. We then discuss the issues that arise as we extend the learning scenario to require collaborative and adversarial learning.

Mixed-initiative settings, where both human and machine are intimately involved in the planning process, pose several challenges for traditional planning frameworks. In [Cox and Veloso 97] we examine the types of unexpected goals that a human may give to the underlying planning system, and thereby how humans change the way planning must be performed. Users may want to achieve goals in terms of actions as well as states, they may specify goals that vary along a dimension of abstraction and specificity, and they may mix both top-level goals and subgoals when describing what they want a plan to do. We show how the Prodigy planning system has met these challenges when integrated with a force deployment tool called ForMAT and describe what opportunities this merger offers a generative planning framework.

Human planners rely strongly on past planning experience to generate new plans. ForMAT is a case-based system that supports human planning through the accumulation of user-build plans, query-driven browsing of past plans, and several plan-functionality-analysis primitives. Prodigy/Analogy is an automated AI planner that combines generative and case-based planning. Stored plans are automated with plan rationale and re-use involves adaptation driven by this rationale. Our system, MI-CBP, integrates ForMAT and Prodigy/Analogy into a realtime, message-passing, mixed-initiative system [Veloso et al 97a]. The main technical approach consists of allowing the user to specify and link objectives that enable the system to capture and reuse plan rationales. We present MI-CBP and its concrete application to the domain of military force deployment planning. The synergetic system increases the planning efficiency of human planners through automated suggestion of similar past plans and plausible plan modifications.

In the past few years, the area of *multiagent systems* has emerged as an active subfield of AI. There is much interest in using machine learning techniques to address the inherent complexity of multiagent systems. Robotic soccer is a particularly good domain for studying multiagent learning. Our approach to using multiagent learning as a tool for building Soccer Server clients involves layering increasingly complex learned behaviors. In [Stone and Veloso 96d] we describe two levels of learned behaviors. First, the clients learn a low-level individual skill that allows them to control the ball effectively. Then, using this learned skill, they learn a higher-level skill that involves multiple players. For each skill, we describe the learning method in detail and report on our extensive empirical testing.

[Stone et al 96] details the hard-coded functionality of the Robotic Soccer agents, the perception model used, and the behaviors acquired by learning.

We describe in [Veloso et al 97b] the architecture of the physical system used in Robotic Soccer and the way actions are layered, building upon each other to create strategic reasoning. We have been using realistic simulation environments to learn basic collaborative strategic procedures. Our ongoing research consists of extending and applying these robust strategic templates to the physical agents. Our approach structures the deliberation and reaction as a layered learning ar-

chitecture. Other efforts have been pursued on applying learning to acquire specific behaviors in different setups. We have chosen to focus on producing a simple, robust design that will enable us to concentrate our efforts.

In [Stone and Veloso 97] we present a novel technique for agent control in a complex multiagent domain based on the confidence factors provided by the C4.5 Decision Tree algorithm. Using Robotic Soccer as an example of such a domain, we incorporated a previously-trained Decision Tree into a full multiagent behavior that is capable of controlling agents throughout an entire game. Along with using Decision Trees for control, this behavior also makes use of the ability to reason about action-execution time to eliminate options that would not have adequate time to be executed successfully. The newly created behavior was tested empirically in game situations.

3.3 Bibliography

[Achim et al. 96]

Achim, S., P. Stone, and M.M. Veloso.

Building a Dedicated Robotic Soccer System.

In *Working Notes of the IROS-96 Workshop on RoboCup*. November, 1996.

[Altmann 96]

Altmann, E.

Episodic Memory for External Information.

Technical Report, School of Computer Science, Carnegie Mellon University, CMU-CS-96-167,

1996.

[Altmann et al. 95]

Altmann, E.M., J.H. Larkin, and B.E. John.

Display navigation by an expert programmer: A preliminary model of memory.

In *Human Factors in Computing Systems: Proceedings of CHI '95*. CHI, 1995.

[Bergmann et al. 96]

Bergmann, R., H. Munoz-Avila, and M.M. Veloso.

General-Purpose Case-Based Planning: Methods and systems.

Kuenstliche Intelligenz 1:22-28, 1996.

In German.

[Blythe 94a]

Blythe, J.

Planning with external events.

In Ramon Lopez de Mantaras and David Poole (editors), *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*. AAAI, July, 1994.

[Blythe 94b]

Blythe, J.

Decision-theoretic subgoalting for planning with external events.

In *Working Notes of the AAAI 1994 Spring Symposium Series, Workshop on Decision-Theoretic Planning*. AAAI, March, 1994.

[Blythe 94c]

Blythe, J.

Probabilistic knowledge of external events in planning.

In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*. AAAI, 1994.

[Blythe 95]

Blythe, J.

AI Planning in Dynamic, Uncertain Domains.

In *Proceedings of the AAAI Spring Symposium on Extending Theories of Action*. Stanford, March, 1995.

[Blythe 96a]

Blythe, J.

Decompositions of Markov Chains for Reasoning about External Change in Planners.

In B. Drabble (editor), *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*. AAAI Press, University of Edinburgh, May, 1996.

[Blythe 96b]

Blythe, J.

Efficient Planning in Dynamic Domains with ADL-like External Events.

1996.

Submitted to the AAAI workshop on Theories of Action, Planning and Control: Bridging the Gap.

[Blythe 96c]

Blythe, J.

The Footprint Principle for Heuristics for Probabilistic Planners.

In Ghallab, M., and A. Milani (editors), *New Directions in AI Planning*, pages 173-185. IOS Press, 1996.

[Blythe and Reilly 93]

Blythe, J. and W.S. Reilly.

Integrating reactive and deliberative planning in a household robot.

In *Working Notes of the AAAI 1993 Fall Symposium Series, Symposium on Instantiating Real-world Agents*. October, 1993.

[Blythe and Veloso 96]

Blythe, J., and M.M. Veloso.

Learning to Improve Uncertainty Handling in a Hybrid Planning System.

In *AAAI Fall Symposium*. AAAI Press, Menlo Park, CA, November, 1996.

To appear.

[Blythe and Veloso 97]

Blythe, J., and M.M. Veloso.

Analogical Replay for Efficient Conditional Planning.

In *Proceedings of the National Conference on Artificial Intelligence*. 1997.

Submitted.

[Blythe et al. 96]

Blythe, J., M.M. Veloso, and L.E. de Souza.

The Prodigy User Interface.

1996.

Submitted to the XIIIth Brazilian Symposium on Artificial Intelligence.

[Borrajó and Veloso 94]

Borrajó, D. and M. Veloso.

Incremental learning of control knowledge for nonlinear problem solving.

In *Proceedings of the European Conference on Machine Learning, ECML-94*. ECML, April, 1994.

[Borrajó and Veloso 96]

Daniel Borrajó and Manuela M. Veloso.

Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans.

Journal of Artificial Intelligence Review, in press, forthcoming 1996.

- [Bowling et al. 96]
Bowling, M., P. Stone, and M.M. Veloso.
Predictive Memory for an Inaccessible Environment.
In *Working Notes of the IROS-96 Workshop on RoboCup*. November, 1996.
- [Cox and Veloso 97]
Cox, M.T., and M.M. Veloso.
Controlling for Unexpected Goals when Planning in a Mixed-Initiative Setting.
In *Proceedings of the National Conference on Artificial Intelligence*. 1997.
Submitted.
- [de Silva 94]
de Silva, R.
Goal-clobbering avoidance in nonlinear planners.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI94, Student Poster Session*. AAAI, August, 1994.
- [de Silva 95]
de Silva, R.
Selectively Delaying Linking Commitments in Partial-order Planners.
In *Proceedings of the Third European Workshop on Planning*. September, 1995.
- [Doorenbos 94]
Doorenbos, R.
Combining left and right unlinking for matching a large number of learned rules.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI, 1994.
An expanded version appears as technical report CMU-CS-94-132.
- [Doorenbos 95]
Doorenbos, R.B.
Production matching for large learning systems.
PhD thesis, Computer Science Department, Carnegie Mellon University, 1995.
- [Doorenbos and Veloso 93]
Doorenbos, R.B. and M.M. Veloso.
Knowledge organization and the utility problem.
In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*. 1993.
- [Fink 95a]
Fink, E.
Design of representation-changing algorithms.
Technical Report CMU-CS-95-120, School of Computer Science, Carnegie Mellon University,
1995.
- [Fink 95b]
Fink, E.
Systematic approach to the design of representation-changing algorithms.
In *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*. 1995.

[Fink 97]

Fink, E.
How to Solve it Automatically: Selection among Problem-Solving Methods.
In Proceedings of the International Conference on Machine Learning. 1997.
Submitted.

[Fink and Blythe 97]

Fink, E., and J. Blythe.
Rasputin: A Complete Bidirectional Planner.
In Proceedings of the International Joint Conference on Artificial Intelligence. 1997.
Submitted.

[Fink and Veloso 94]

Fink, E. and M.M. Veloso.
Prodigy planning algorithms.
Technical Report CMU-CS-94-123, Computer Science Department, Carnegie Mellon University,
1994.

[Fink and Veloso 95]

Fink, E., and M.M. Veloso.
Formalizing the Prodigy Planning Algorithm.
In Proceedings of the Third European Workshop on Planning. September, 1995.

[Fink and Veloso 96]

Fink, E., and M.M. Veloso.
Formalizing the Prodigy Planning Algorithm.
In M. Ghallab and A. Milani (editors), New Directions in AI Planning, pages 261--272. IOS Press, 1996.
An earlier, extended version is available as technical report CMU-CS-94-123, 1994.

[Fink and Yang 94a]

Fink, E. and Q. Yang.
Search reduction in planning with primary effects.
In Proceedings of the Workshop on Theory Reformulation and Abstraction. 1994.

[Fink and Yang 94b]

Fink, E. and Q. Yang.
Automatically selecting and using primary effects in planning: theory and experiments.
Technical Report CMU-CS-94-206, Computer Science Department, Carnegie Mellon University,
1994.

[Fink and Yang 95]

Fink, E., and Q. Yang.
Planning with primary effects: Experiments and analysis.
In Proceedings of the International Joint Conference on Artificial Intelligence. 1995.

[Fink and Yang 96]

Fink, E., and Q. Yang.
Automatically Selecting and Using Primary Effects in Planning: Theory and Experiments.
AI Journal, 1996.
To appear.

[Gil and Perez 94]

Gil, Y. and M.A. Perez.

Applying a general-purpose planning and learning architecture to process planning.

In *Proceedings of the AAAI 1994 Fall Symposium Series, Symposium on Planning and Learning: On to Real Applications*. AAAI, November, 1994.

[Green and Lehman 96a]

Green, N., and J.F. Lehman.

An Approach to Compiling Knowledge for Dialogue Generation and Interpretation.

In *Proceedings of the Thirty-fourth Annual Meeting of the Association for Computational Linguistics*. 1996.

[Green and Lehman 96b]

Green, N., and J.F. Lehman.

Goals for Future Computational Models of Conversational Implicature.

In *Proceedings of the 1996 AAAI Spring Symposium on Computational Implicature*. 1996.

[Green and Lehman 96c]

Green, N., and J.F. Lehman.

Comparing Agent Modeling for Language and Action.

In *Agent Modeling: Papers from the 1996 AAAI Workshop*, Technical Report WS-96-02. 1996.

[Green and Lehman 96d]

Green, N., and J.F. Lehman.

Compiling Knowledge for Dialogue Generation and Interpretation.

Technical Report , School of Computer Science, Carnegie Mellon University, CMU-CS-96-175, 1996.

[Green and Lehman 97]

Green, N., and J.F. Lehman.

An Integrated Architecture for Discourse: Generation, Interpretation, and Recipe Acquisition.

In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. 1997. Submitted.

[Haigh and Shewchuk 94]

Haigh, K.Z. and J.R. Shewchuk.

Geometric similarity metrics for case-based reasoning.

In *Case-Based Reasoning: Working Notes from the AAAI-94 Workshop*. AAAI, August, 1994.

[Haigh and Veloso 95]

Haigh, K.Z., and M.M. Veloso.

Route Planning by Analogy.

In Manuela Veloso and Agnar Aamodt (editors), *Case-Based Reasoning Research and Development, Proceedings of ICCBR-95*, pages 169-180. Springer-Verlag, Sismbra, Portugal, October, 1995.

Available at <http://www.cs.cmu.edu/khaigh/papers.html>.

[Haigh and Veloso 96a]

Haigh, K.Z., and M.M. Veloso.

Planning with Multiple Goals for Robot Execution.

In *Proceedings of the AAAI Fall Symposium "Plan Execution: Problems and Issues"*. AAAI Press, Menlo Park, CA, November, 1996.

To appear.

[Haigh and Veloso 96b]

Haigh, K.Z., and M.M. Veloso.

Interleaving Planning and Robot Execution for Asynchronous User Requests.

In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. November, 1996.

To Appear. Also appeared in Proceedings of the AAAI Spring Symposium, "Planning with Incomplete Information for Robot Problems", March, 1996.

[Haigh and Veloso 96c]

Haigh, K.Z., and M.M. Veloso.

Using Perception Information for Robot Planning and Execution.

In *Proceedings of the AAAI Workshop "Intelligent Adaptive Agents"*, pages 23-32. AAAI Press, Menlo Park, CA, August, 1996.

[Haigh and Veloso 97a]

Haigh, K.Z., and M.M. Veloso.

High-Level Planning and Low-Level Execution: Towards a Complete Robotic Agent.

In *Proceedings of the First International Conference on Autonomous Agents*. 1997.

[Haigh and Veloso 97b]

Haigh, K.Z., and M.M. Veloso.

Identifying and Using Learning Opportunities From Robot Execution.

In *Proceedings of the International Conference on Machine Learning*. 1997.

Submitted.

[Haigh et al. 94]

Haigh, K.Z., J.R. Shewchuk, and M.M. Veloso.

Route planning and learning from execution.

In *Working notes from the AAAI Fall Symposium on Planning and Learning: On to Real Applications*. AAAI, November, 1994.

[Haigh et al. 96]

Haigh, K.Z., M.M. Veloso, and J.R. Shewchuk.

Exploiting Domain Geometry in Analogical Route Planning.

Journal of Experimental and Theoretical Artificial Intelligence, 1996.

To appear.

[Laird et al. 95]

Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. S.,

Rosenbloom, P. S., Rubinoff, R., Schwamb, K. B., and Tambe, M.

Simulated Intelligent Forces for Air: The Soar/IFOR Project 1995.

In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, pages 27-36. 1995.

- [Lehman 94]
Lehman, J.F.
Towards the essential nature of statistical knowledge in sense resolution.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI. 1994.
- [Lehman et al. 95]
Lehman, J. F., Van Dyke, J., and Rubinoff, R.
Natural Language Processing for IFORS: Comprehension and Generation in the Air Combat Domain.
In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, pages 115-123. 1995.
- [Lewis 93]
Lewis, R. L.
An architecturally-based theory of human sentence comprehension.
PhD thesis, Computer Science Department, Carnegie Mellon University, 1993.
Also available as Technical Report CMU-CS-93-226.
- [Lonsdale 96a]
Lonsdale, D.
An Architectural Theory for Simultaneous Interpretation.
In *Proceedings of the Canadian Association of Applied Linguistics*. 1996.
- [Lonsdale 96b]
Lonsdale, D.
Cognitive Modelling of Simultaneous Interpretation.
In *Proceedings of the Colloque Etudiant de Linguistique Informatique de Montreal*. 1996.
- [Lonsdale 97]
Lonsdale, D.
Modelling SI Cognition: A UTC-Based Approach.
*Interpreting: International Journal of Research and Practice in Interpreting*2, 1997.
- [Melis and Veloso 94]
Melis, E. and M. Veloso.
Analogy makes proofs feasible.
In *Preprints of the AAAI Workshop on Case-based Reasoning*. AAAI, August, 1994.
- [Nelson et al. 94a]
Nelson, G., J.F. Lehman, and B.E. John.
Experiences in interruptible language processing.
In *Proceedings of the 1994 AAAI Spring Symposium on Active NLP*. 1994.
- [Nelson et al. 94b]
Nelson, G., J.F. Lehman, and B.E. John.
Integrating cognitive capabilities in a real-time task.
In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. CSS, 1994.

[Newell et al. 91]

Newell, A., G.R. Yost, J.E. Laird, P.S. Rosenbloom, and E. Altmann.
Formulating the problem space computational model.
In R.F. Rashid (editor), *Carnegie Mellon Computer Science: A 25-Year Commemorative*.
ACM-Press: Addison-Wesley, 1991.

[Pelton and Lehman 94]

Pelton, G. and J.F. Lehman.
The breakdown of operators when interacting with the external world.
Technical Report CMU-CS-94-121, Computer Science Department, Carnegie Mellon University,
February, 1994.

[Pelton and Lehman 95]

Pelton, G. A. and Lehman, J. F.
Everyday Believability.
Technical Report, School of Computer Science, Carnegie Mellon University, CMU-
CS-95-133,
1995.

[Pelton and Lehman 96]

Pelton, G. A., and J.F. Lehman.
Being effective when suspending goals.
In *Proceedings of AAAI Workshop on Plan Execution: Problems and Issues*. 1996.

[Perez 94a]

Perez, M.A.
Learning quality-enhancing control knowledge.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI94, Student Poster Session*. AAAI, August, 1994.

[Perez 94b]

Perez, M.A.
The goal is to generate better plans.
In *Working Notes of the AAAI 1994 Spring Symposium Series, Workshop on Goal-Driven Learning*. AAAI, March, 1994.

[Perez 94c]

Perez, M.A.
Aprendizaje automatico en los sistemas de planificacion.
In *Invited Lecture at the VIth International Symposium, Systems: Transition towards Automation, Universidad Autonoma de Coahuila, Mexico*. March, 1994.

[Perez 95]

Perez, M.A.
Learning Search Control Knowledge to Improve Plan Quality.
PhD thesis, Computer Science Department, Carnegie Mellon University, 1995.
Available as technical report CMU-CS-95-175.

[Perez 96]

Perez, M.A.

Learning from a Domain Expert to Generate Good Plans.

In *Proceedings of the AAAI Spring Symposium "Acquisition, Learning and Demonstration: Automating Tasks for Users"*, pages 110-116. AAAI Press, March, 1996.

[Perez and Carbonell 94]

Perez, M.A. and J.G. Carbonell.

Control knowledge to improve plan quality.

In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*. AIPS, June, 1994.

[Perez et al. 96]

Perez, A., J. Blythe, and M.M. Veloso.

Comparing the Representational Power of Operator-Based and HTN Planners.

In *Proceedings of AAAI '96*. 1996.

Submitted.

[Rubinoff and Lehman 94]

Rubinoff, R. and J.F. Lehman.

Real-time natural language generation in NL-Soar.

In *Proceedings of the 7th International Workshop on Natural Language Generation*. 1994.

[Smirnov and Veloso 96]

Smirnov, Y., and M.M. Veloso.

Efficiency Competition through Representation Changes: Pigeonhole Principle versus Linear Programming Relaxation.

In *Proceedings of KR'96, the Fifth International Conference on Principles of Knowledge Representation and Reasoning*. November, 1996.

[Smirnov and Veloso 97]

Smirnov, Y., and M.M. Veloso.

GSAT: A Navigational Approach.

In *Proceedings of the National Conference on Artificial Intelligence*. 1997.

Submitted.

[Smirnov et al. 96]

Smirnov, Y., S. Koenig, M.M. Veloso, and R. Simmons.

Efficient Goal-Directed Exploration.

In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 292-297. August, 1996.

[Smith and Lehman 97]

Smith, M., and J.F. Lehman.

Natural Language Communication by Intelligent Agents in an Air Combat Domain.

In *Proceedings of the 1997 Symposium on Aviation Communication, Embry-Riddle Aeronautical University*. 1997.

[Soar 94]

The Soar Group.

The Soar Project.

In *Proceedings of Soar Workshop XIII*. Ohio State University, 1994.

[Souza and Veloso 96a]

Souza, L.E., and M.M. Veloso.

AI Planning for a Supervisory Control System.

In *Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics*. China, October, 1996.

[Souza and Veloso 96b]

Souza, L.E., and M.M. Veloso.

Acquisition of Flexible Planning Knowledge from Means-ends Models for Industrial Processes.

IEEE Transactions on Knowledge and Data Engineering (TKDE), June, 1996.

[Souza and Veloso 96c]

Souza, L.E., and M.M. Veloso.

Flexible Planning Knowledge Acquisition for Industrial Processes.

In *Proceedings of the First International Conference on Industrial Engineering Applications and Practice*. Houston, TX, December, 1996.

[Stone and Veloso 94a]

Stone, P. and M. Veloso.

The need for different domain-independent heuristics.

In *Proceedings of the Second International Conference on AI Planning Systems*. June, 1994.

[Stone and Veloso 94b]

Stone, P. and M.M. Veloso.

Learning to solve complex planning problems finding useful auxiliary problems.

In *Technical Report of the AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications*. AAAI, November, 1994.

[Stone and Veloso 95]

Stone, P., and M.M. Veloso.

User-guided Interleaving of Planning and Execution.

In *Proceedings AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, 1995*. September, 1995.

[Stone and Veloso 96a]

Stone, P., and M.M. Veloso.

User-guided Interleaving of Planning and Execution.

In M. Ghallab and A. Milani (editors), *New Directions in AI Planning*, pages 103--112. IOS Press, 1996.

Proceedings of the European Workshop on Planning.

[Stone and Veloso 96b]

Stone, P., and M.M. Veloso.

Multiagent Systems: A survey from a machine learning perspective.

IEEE Transactions on Knowledge and Data Engineering (TKDE), June, 1996.

[Stone and Veloso 96c]

Stone, P., and M.M. Veloso.

Towards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer.

To appear in *International Journal of Human-Computer Systems (IJHCS)*, 1996.

- [Stone and Veloso 96d]
Stone, P., and M.M. Veloso.
Using Machine Learning in the Soccer Server.
In *Proceedings of the IROS-96 Workshop on RoboCup*. 1996.
- [Stone and Veloso 97]
Stone, P., and M.M. Veloso.
Using Decision Tree Confidence Factors for Agent Control.
In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1997.
Submitted.
- [Stone et al 96]
Stone, P., M.M. Veloso, and S. Achim.
Collaboration and Learning in Robotic Soccer.
In *Proceedings of the Micro-Robot World Cup Soccer Tournament*. IEEE Robotics and Automation Society, 1996.
- [Van Dyke and Lehman 97]
Van Dyke, J., and J.F. Lehman.
A Process Model of Learning Definiteness in a Foreign Language.
In *Proceedings of the Nineteenth Cognitive Science Society*. 1997.
Submitted.
- [Veloso 96a]
David B. Leake (editor).
Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes.
AAAI Press/The MIT Press, 1996.
Book version of the paper presented in the Twelfth National Conference on Artificial Intelligence, AAAI Press, 1994.
- [Veloso 96b]
Veloso, M.M.
Towards Mixed-Initiative Rationale-Supported Planning.
In A. Tate (editor), *Advanced Planning Technology*, pages 277-282. AAAI Press, 1996.
- [Veloso and Aamodt 96]
Veloso, M.M., and A. Aamodt (editor).
Case-Based Reasoning Research and Development.
Springer Verlag, 1996.
- [Veloso and Blythe 94]
Veloso, M. and Blythe, J.
Linkability: Examining causal link commitments in partial-order planning.
In *Proceedings of the Second International Conference on AI Planning Systems*. June, 1994.
- [Veloso and Stone 95]
Veloso, M.M. and P. Stone.
FLECS: planning with a flexible commitment strategy.
Journal of Artificial Intelligence Research, 1995.

[Veloso et al 97a]

Veloso, M.M., A.M. Mulvehill, and M.T. Cox.

Rationale-Supported Mixed-Initiative Case-Based planning.

In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*. 1997.
Submitted.

[Veloso et al 97b]

Veloso, M.M., P. Stone, S. Achim.

A Layered Approach for an Autonomous Robotic Soccer System.

In *Proceedings of the First International Conference on Autonomous Agents*. 1997.

[Veloso et al. 95]

Veloso, M.M., J.G. Carbonell, M.A. Perez, D. Borrajo, E. Fink, and J. Blythe.

Integrating planning and learning: The Prodigy architecture.

Journal of Experimental and Theoretical Artificial Intelligence 7(1):81-120, 1995.

[Wang 94a]

Wang, X.

Learning planning operators by observation and practice.

In *Proceedings of the Second International Conference on AI Planning Systems, AIPS-94*.
AIPS, June, 1994.

[Wang 94b]

Wang, X.

Learning by observation and practice: A framework for automatic acquisition of planning operators.

In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI94, Student Poster Session*. AAAI, August, 1994.

[Wang 95]

Wang, X.

Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition.

In *Proceedings of the 12th International Conference on Machine Learning*. 1995.

[Wang 96]

Wang, X.

Planning While Learning Operators.

In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*. University of Edinburgh, 1996.

[Wang and Carbonell 94]

Wang, X. and J.G. Carbonell.

Learning by observation and practice: towards real applications of planning systems.

In *AAAI-94 Fall Symposium Series: Planning and Learning: On to Real Applications*. AAAI, 1994.

[Wang and Veloso 94]

Wang, X. and M. Veloso.

Learning planning knowledge by observation and practice.

In *Proceedings of the ARPA Planning Workshop*. ARPA, February, 1994.

4. Machine Learning in Large Scale Software Environments

Our research in Machine Learning sought to develop and demonstrate useful learning capabilities within significantly sized software systems, such as autonomous robots. This work requires scaling up previous learning algorithms by (1) extending them to handle more complex learning tasks while still using realistic amounts of training data and (2) integrating learning algorithms into domain-specific software architectures for robot planning and control, so that they apply uniformly to applications within the architecture.

We have taken the approach of developing and demonstrating learning capabilities for mobile robots and other large software systems. Our three main thrusts are: (1) developing and characterizing specific algorithms for machine learning, (2) developing robot systems on an underlying general architecture called TCA, which is in use by a variety of fielded robot systems at Carnegie Mellon and elsewhere, and (3) integrating these learning techniques within TCA so that they are invoked automatically by the architecture as part of routine robot operation.

4.1 Reliable indoor navigation

We performed extensive experiments in reliable indoor navigation. We developed passive, unsupervised, realtime learning techniques to refine probabilistic models of the robot's sensors and the environment. We developed algorithms that learn to classify objects and estimate 3D position using single-frame color images. We have been working on combining RAPs and TCA, transforming RAP expressions into code that utilizes the TCA library of control constructs. We also developed and evaluated a new learning algorithm that enables a mobile robot to discover landmarks on its own and pursued our research on "lifelong learning" algorithms for accurately learning to recognize entire families of objects. We improved our previous methods for learning metric, two-dimensional robot maps and invented a new method for learning topological maps, which method was found to reduce the motion planning time by several orders of magnitude. A prototype cleaning robot developed in our lab tied for first prize at the AAAI mobile robot competition, and a major commercial robot manufacturer now distributes Carnegie Mellon-developed software as its sole navigation software.

We had previously developed a navigation system that uses partially observable Markov decision process (POMDP) models to track robot position and control the robot. The probabilistic information allows the robot to navigate more reliably and facilitates learning to improve the robot's models and to detect failures. We worked to integrate our probabilistic navigation and landmark discovery work, in order to enable the robot to learn new landmarks in exactly those areas where it robot finds it has trouble navigating. We did work in visual learning, including finding doors, finding landmarks, recognizing people, classifying objects and estimating 3D position. We also devised new methods for learning topological maps for indoor navigation.

4.2 Theoretical results for reinforcement learning

One promising approach to control learning, currently being explored by many research groups, is *reinforcement learning*. The key idea of reinforcement learning methods such as Q-learning, is that they acquire a successful control policy by experimenting in their environment and observing the consequences. More specifically, training data consists of a delayed reward signal that indicates when some sequence of actions has lead to a desirable state. Reinforcement learn-

ing is relevant to learning for mobile robots and for learning to optimize manufacturing processes.

Previous research in Q-learning has shown theoretically that it can converge to optimal control policies for Markov processes under certain circumstances. A major restriction in these theoretical results is that they require that the learner acquire an evaluation function *represented by a complete table, with one entry for each possible state-action pair*. In other words, these results do not apply to Q-learning approaches that use techniques other than rote learning. In contrast experimental results have shown that, when based on function approximators such as Back-propagation and EBNN, Q-learning sometimes converges and sometime does not. A better theoretical characterization of Q-learning is needed in order to understand when and how it might be used in practical problems.

We developed new theoretical results relevant to understanding the convergence of reinforcement learning using function approximators. In particular we showed that for parallel-update, offline dynamic programming (a problem closely related to Q-learning), the algorithm converges for both rote learning and for certain types of generalizing function approximators. This is the first published proof about convergence of this class of algorithms, using generalizing function approximators rather than rote learning. It shows, for example, that nearest-neighbor approaches will converge, whereas neural network approaches will not converge in general. This work is described in more detail in a paper submitted to the International Conference on Machine Learning.

4.3 Probabilistic navigation

A landmark-based navigation scheme for our indoor mobile robot that uses reactive monitors and exception handlers to achieve reliable behavior. Although its navigation characteristics were good, further experience showed that the robot would occasionally get lost, and we determined that incorporating learned feature detectors (a major goal of this research) would not be straightforward.

We developed an alternative navigation scheme that uses probabilistic, partially-observable Markov models to track the robot's position and to indicate which actions to take. The nodes of the Markov model represent discrete positions and orientations of the robot, and the navigation scheme uses sensor readings (currently sonar and dead-reckoning) and Bayes' rule to update the probabilities that the robot is at each location. To actually navigate, a path planner associates actions with each Markov node, and, whenever the probabilities are updated, the action with the highest total "probability" mass is taken.

This approach has several advantages over previous navigation schemes, including our own landmark-based navigation. First, it is easy to add new (learned) feature detectors, merely by providing a table of conditional observation probabilities for each Markov node. Second, our formulation explicitly encodes uncertainty in the lengths of corridors, so that accurate metric maps are not needed. Third, the representation of positional uncertainty is better suited to indoor navigation than, say, Kalman filters, since the Markov models can represent more general probability distributions (e.g., "the robot is in front of either door101 or door37"). Finally, the robot does not get lost nearly so often, since position tracking is much more robust.

4.4 Learning robot action effects

One key type of knowledge for robotic applications is that of the effects of robot actions. Robot planning approaches that search through the space of possible robot actions rely crucially on such knowledge of action preconditions and effects. However, most experimental work in robotics encounters difficulties in modelling the true preconditions and effects of actions. We have begun exploring the feasibility of learning such action models from experience.

In one set of experiments, neural network backpropagation was used to learn the effects of several potential Xavier mobile robot field navigation actions. For each of these actions, a neural network learned to predict the values of future sensor readings (after performing the action) based on the current sensor readings (before performing the action). Sensor readings include sonar range readings, laser range values, and the position of a target object within the visual field of a color camera. Each action corresponds to applying the potential field navigation procedure for approximately two seconds, attempting to reach a particular target position while treating observed obstacles as repulsive forces. These actions are quite difficult to model due to the unpredictable effects of specific types of obstacles on the actual robot trajectory and to sensor noise. Approximately 3000 training examples of these action effects were collected by allowing the robot to roam through a cluttered laboratory. The learned action models were able to characterize typical effects of the actions, including the effects of many types of obstacles [Mitchell and Thrun 94]. The approximate learned action models are probably too inaccurate to be used in straightforward planning algorithms, though future work will explore ways in which such approximate action models can be useful. Despite this, we have found these approximate action models are quite useful for guiding subsequent learning of robot control strategies.

4.5 Learning road features for autonomous driving

In collaboration with Carnegie Mellon's Autonomous Land Vehicle Navlab effort, we explored the feasibility of supervised learning of useful road feature recognizers (e.g., road edge, center line). The goal of this work is to learn efficient, reliable feature recognizers that are of use for autonomous driving.

The key result during this period, reported in [Thorpe et al. 94], was the use of neural network backpropagation to learn useful feature detectors for road edges and centerlines. These learned features:

- are more reliable than the hand-crafted feature recognizers used in the current YARF system, and
- were learned automatically from training data generated by the current YARF system.

Somewhat surprisingly, the neural networks learned from training data provided by YARF are more reliable than the YARF recognizers. We believe this may be a side effect of the inductive bias of neural networks (which tends to interpolate smoothly among training examples), coupled with signal and noise characteristics of data for this task (where many image pixels can provide clues about the feature, but no single pixel can be trusted). While we still do not understand the range of tasks for which this phenomenon may hold, it appears that at least in this case, the learned features will be useful.

4.6 EBNN learning

The EBNN learning algorithm has been developed as a learning technique that scales up to more complex learning tasks than can purely inductive methods such as neural network Backpropagation. EBNN is a more powerful learning method because it uses knowledge previously learned by the robot to augment the training data available for the new learning task. The TCA architecture has been used as the basis for a mobile navigating robot, incorporating a robust planning technique based on Partially Observable Markov Models. An initial design has been developed for incorporating EBNN into TCA as the next step in this research.

We applied the EBNN learning algorithm more thoroughly to a mobile robot perceptual task, by conducting a thorough experimental study based on the Xavier mobile robot. This study, described in [O'Sullivan et al. 95], explores several different ways in which prior knowledge could be used to explain new observations, thus guiding learning. The results reinforce our initial assessment that EBNN provides a more effective learning mechanism than Backpropagation in cases where the robot must learn a new task in a familiar environment. It also provided experimental evidence that EBNN is robust to errors in the previously learned action models — its performance degrades gracefully with increasing error in the action models, in the limit approaching the performance of purely inductive Backpropagation. We have extended the initial studies reported in this paper by comparing the performance of EBNN to an alternative algorithm for utilizing prior knowledge, called KBANN (developed at the University of Wisconsin). EBNN and KBANN appear to offer complementary advantages in terms of their robustness to errors in prior knowledge and in the way they utilize this knowledge to influence learned network weights. We have therefore begun to explore ways of combining these two approaches into a third algorithm that would subsume both. We conducted initial experiments, but the work has not, during this reporting period, led to publishable results.

EBNN was also applied to visual object recognition based on a single image. Object appearance varies because of translation and rotation of the object and because of differences in lighting conditions. When presented training data for several types of objects, the system learned an invariance function — a mapping ($\text{Image1} \times \text{Image2} \rightarrow \{T, F\}$) whose value is T only if Image1 and Image2 are images of the same object. This *invariance mapping* approximately characterizes the notion of "equivalent object," relative to the typical changes in lighting, translation, and rotation within our domain. During the past six months we have completed the set of preliminary experiments begun earlier, completed the final version of a technical paper on this work [Thrun and Mitchell 95], and presented this paper at IJCAI95.

We completed our experimental analysis of EBNN and a large-scale comparison of different methods (all developed at Carnegie Mellon) that transfer knowledge across multiple learning tasks.

We determined that transferring knowledge is more important than a careful selection of the particular learning algorithm. In our study we found that all algorithms that transfer knowledge generalize significantly more accurately from less data than the best algorithms that do not transfer knowledge [Thrun 96a, Thrun 96b].

We completed our experiments with the EBNN algorithm, demonstrating that by using previously learned knowledge it is able to significantly outperform purely inductive learning algorithms such as neural network backpropagation. Based on this experience we began exploring

a variety of new algorithms for using prior knowledge, as detailed below. In addition we developed new approaches to learning maps for navigation and began design of a "lifelong learning" architecture intended to accumulate knowledge over a long period of robot operation.

4.7 Improving techniques

4.7.1 Navigation

We extended our POMDP model of robot navigation to better model foyers, rooms, and corridors of different widths. This enables the robot to travel to more places, more reliably, and enables us to run the robot in more varied environments.

To test the reliability of our probabilistic navigation algorithms, we created a Web-based interface (<http://www.cs.cmu.edu/~xavier>). The interface enables people worldwide to see what the robot sees and to command it to go to various offices in our building. A task scheduler handles the myriad requests, deciding where to send the robot. Since last December, the robot has operated nearly every other day and has traveled over 70 kilometers, performing over 2000 jobs, and successfully reaching the commanded room nearly 95% of the time.

We developed a real-time learning algorithm that refines the probability estimates of our POMDP models. This algorithm can learn such things as the probability of observing features, given that they occur in the environment, the dead-reckoning characteristics of the robot, and the (perceived) lengths of corridors. The algorithm runs in background mode, as the robot is performing its tasks, and is unsupervised (needs no "ground truth"). We have tested the algorithm offline, worked to integrate it into our current navigation system.

4.7.2 Learning visual features

We conducted several experiments in learning visual features. In one, we trained a neural network to recognize when it was in front of closed and open doors. We now use that learned network on a daily basis in our WWW-based navigation system to allow the robot to place itself in front of doorways, based on its own visual information. We also worked on having the robot use this network to learn the angular direction to the door, an ability that would significantly reduce the time needed to perform the servoing. We also implemented a memory-based learning technique to locate open doorways and estimate the robot's distance and angle to the doorway.

Another experiment in visual learning integrated eigenvector-based object classification (developed by Shree Nayar at Columbia) and color-histogram feature detection (developed by Mike Swain at U. Chicago) to classify a variety of objects and estimate their 3D positions using only a single color image. The combined algorithm runs more reliably than the color histogram method alone and is significantly faster than the original, pixel-based, object classification algorithm. This combined algorithm will be applied to our ongoing work in mobile manipulation.

4.7.3 Autonomously-discovered landmarks

We developed and tested an algorithm that enables a robot to discover its own landmarks. This procedure differs from previously published algorithms in that no human is required to define appropriate landmarks. While "discovering" landmarks, the robot trains neural networks for their detection. Our initial tests suggest that the new approach reduces the error in localization by a factor of 3 to 20 when compared with our best previous visual method.

We worked to integrate learning visual landmarks with our POMDP models to automatically learn new landmarks in areas of the building where the robot has trouble recognizing where it is. The idea is to notice when the entropy of the probability distribution gets high, and then learn features in that specific area that minimize distance error and/or entropy. These feature detectors can then be selectively activated whenever the robot believes that it may be approaching that area.

4.7.4 Obstacle avoidance

We developed a realtime, local-obstacle-avoidance method that considers the robot dynamics, enabling the robot to travel significantly faster than previously (60-90 cm/s, depending on the speed of the proximity sensors) and more smoothly. This algorithm is now the primary obstacle avoidance technique used by our robots. It is also being distributed commercially by a leading mobile robot manufacturer, Real World Interfaces Inc. (Jaffrey, New Hampshire).

4.7.5 Robot control software

RAPs (developed by Jim Firby at U. Chicago) and TCA (Task Control Architecture, developed by Simmons at Carnegie Mellon) are two of the more popular task-level-robot control architectures. One advantage of RAPs is a nice syntax for expressing control strategies. An advantage of TCA is its distributed, concurrent nature. We worked to combine RAPs and TCA. The idea was to transform RAPs expressions into code that utilizes the TCA library of control constructs. We have succeeded in distributing the RAPs symbolic data base, using TCA's network communications mechanisms, and have implemented a subset of RAPs expressions using TCA control constructs. Our TCA robot control architecture has been selected for use in NASA's New Millenium project.

We improved our methods for autonomous robot exploration and learning two-dimensional occupancy maps by incorporating laser rangefinder data. Our current software is able to map reliably areas up to size 60 by 20 meters, that were impossible with our previous software. Our new algorithm extracts topological maps from our learned occupancy maps. By using topological maps, we were able to reduce the complexity of motion planning by three to six orders of magnitude, incurring an actual performance loss (detours) of less than 4%. As a result, for reasonably sized indoor environments, we can now efficiently pre-compute all motion plans, so that no further planning is necessary during everyday robot operation. Our map learning and planning software is now being distributed along with robots manufactured by Real World Interfaces, Inc., as their sole navigation software.

4.7.6 Testbed

We built a mobile, rotating-brush-welding robot that employs a stripped-down version of our best map learning, collision avoidance, and position control methods. The robot came in first at the preliminary trials and tied for first place at finals of the 1996 AAAI mobile robot competition (both in the category "clean-up a tennis court.")

Finally, we developed a new "lifelong" learning method (TC) that learns to recognize landmarks/people/objects and locations. Unlike our previous methods, this new technique can selectively build on related, previously learned knowledge to guide generalization. Empirically, we demonstrated a significant reduction in sample complexity when knowledge was transferred from previous, similar learning tasks.

4.8 Bibliography

[Fox et al. 96a]

Fox, D., W. Burgard, and S. Thrun.
The Dynamic Window Approach to Collision Avoidance.
IEEE Robotics and Automation, 1996.
To appear.

[Fox et al. 96b]

Fox, D., W. Burgard, and S. Thrun.
Controlling Synchro-drive Robots with the Dynamic Window Approach to Collision Avoidance.
In *Proceedings of the IROS, 1996*. IROS, 1996.

[Gordon 95]

Gordon, G.
Stable Function Approximation in Value Iteration.
In *Proceedings of the 12th International Conference on Machine Learning*. July, 1995.

[Koenig and Simmons 96a]

Koenig, S., and R.G. Simmons.
The Effect of Representation and Knowledge on Goal-Directed Exploration with Reinforcement-Learning Algorithms.
Machine Learning Journal(22):227-250, 1996.

[Koenig and Simmons 96b]

Koenig, S., and R.G. Simmons.
Unsupervised Learning of Probabilistic Models for Robot Navigation.
In *Proceedings of ICRA 96*. ICRA, 1996.

[Koenig and Simmons 96c]

Koenig, S., and R.G. Simmons.
Passive Distance Learning for Robot Navigation.
In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages -"266-274". ICML, 1996.

[Koenig et al. 96]

Koenig, S., R. Goodwin, and R.G. Simmons.
Robot Navigation with Markov Models: A Framework for Path Planning and Learning with Limited Computational Resources.
In *Reasoning with Uncertainty in Robotics*. Springer, 1996.

[Mitchell and Thrun 94]

Mitchell, T.M. and S. Thrun.
Learning analytically and inductively.
In Steier, D. and T. Mitchell (editors), *Mind Matters*. Lawrence Erlbaum Associates, 1994.

[Mitchell et al. 94]

Mitchell, T.M., J. O'Sullivan, and S. Thrun.
Explanation-based learning for mobile robot perception.
In *Proceedings of the Workshop on Learning Robots*. July, 1994.

[O'Sullivan et al. 95]

O'Sullivan, J., T.M. Mitchell, and S. Thrun.

Explanation based learning for mobile robot perception.

In K. Ikeuchi and M.M. Veloso (editor), *Symbolic Visual Learning*. Oxford University Press, 1995.

[Simmons 93]

Simmons, R.

Expectation-based behavior.

In *Proceedings of the International Symposium of Robotics Research*. ISRR, 1993.

[Simmons 94]

Simmons, R.

Becoming increasingly reliable.

In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*. AIPS, June, 1994.

[Simmons 95]

Simmons, R.

Towards Reliable Autonomous Agents.

In *Proceedings of the AAAI Spring Symposium on Software Architectures*. AAAI, March, 1995.

[Simmons 96a]

Simmons, R.

Where in the World is Xavier, the Robot?

In *SPIE International Technical Working Group on Robotics and Machine Perception*. 1996.

[Simmons 96b]

Simmons, R.

The Curvature Velocity Method for Local Obstacle Avoidance.

In *Proceedings of ICRA 96*. IEEE, 1996.

[Simmons and Koenig 95]

Simmons, R., and S. Koenig.

Probabilistic Navigation in Partially Observable Environments.

In *Proceedings of the International Joint Conference on AI*. August, 1995.

[Thorpe et al. 94]

Thorpe, C., C. Athanassiou, J. Kay, T. Mitchell, and D. Pomerleau.

Machine learning and human interface for the CMU Navlab.

In *Proceedings of the 6th International Symposium on Robotics Research*. 1994.

[Thrun 96a]

Thrun, S.

Explanation-Based Neural Network Learning: A Lifelong Learning Approach.

Kluwer Academic Publishers, 1996.

[Thrun 96b]

Thrun, S.

An Approach to Learning Mobile Robot Navigation.

Robotics and Autonomous Systems(15):301-319, 1996.

[Thrun 96c]

Thrun, S.

Is Learning the n-th Thing any Easier Than Learning The First?

In D. Touretzky and M. Mozer (editor), *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 1996.

[Thrun and Buecken 96]

Thrun, S., and A. Buecken.

Integrating Grid-Based and Topological Maps for Mobile Robot Navigation.

In *Proceedings of the AAAI, 1996*. AAAI, 1996.

[Thrun and Mitchell 95]

Thrun, S., and T.M. Mitchell.

Learning One More Thing.

In *Proceedings of the International Joint Conference on AI*. August, 1995.

Also appeared as Technical Report CMU-CS-94-184, September, 1994.

[Thrun and O'Sullivan 96a]

Thrun, S., and J. O'Sullivan.

Discovering Structure in Multiple Learning Tasks: The TC Algorithm.

In *Proceedings of the 13th International Conference on Machine Learning ICML-96*. ICML, 1996.

[Thrun and O'Sullivan 96b]

Thrun, S., and J. O'Sullivan.

Learning More From Less Data: Experiment in Lifelong Learning.

In *Seminar Digest*. IEE, 1996.

5. Specification, Verification, and Program Development

Our research goals in this area include developing:

- Methodologies and tools that allow digital system designers to specify desired system behavior formally and to test their designs for adherence to the specification,
- A combined language, programming environment, and runtime support system that provides the power of parallel supercomputers to nonspecialists.

5.1 Parallel computing

Our research goal in this area is to develop techniques for expressing programs that can map efficiently onto a variety of different parallel machines. We believe that by developing an appropriate set of linguistic abstractions, these programs can be expressed in a manner that is independent of the machine architecture or detailed configuration. Our research proceeds on several fronts, including language design, compilation techniques, programming environments, parallel algorithms, and applications.

In recent years the parallel processing community has made major steps in simplifying the use of a variety of high-performance parallel and vector computers by supplying languages that port among these machines. Such languages include High Performance Fortran, C*, NESL, and UC. These languages are approaching the point where it will be possible to write code once and then run it on a variety of different parallel processors with good runtime efficiency. The languages by themselves, however, do not go far enough to make the machines accessible to nonexperts. To significantly increase the utility of high-performance parallel processors, the community needs to: (1) supply a simple and uniform interface for accessing parallel machines and (2) reduce the time required by users to prototype new parallel algorithms. To address these issues, we are working on an environment and associated language that:

- Allows users to access transparently remote parallel processors as servers,
- Supplies a common debugging environment across diverse machines,
- Allows users to interactively run interactively on parallel machines from their local machine and get immediate results,
- Makes it easy to extend and modify existing libraries,
- Supplies common tools for analyzing running times and optimizing code.

By reducing the overhead associated with learning new environments for different machines, these features go beyond the convenience provided by a portable parallel language. They would make parallel processors significantly more attractive to users who have no particular interest in parallel computing but who would greatly benefit from such computational capabilities.

The Scandal project is developing techniques for expressing programs that can map efficiently onto a variety of parallel machine architectures without changes to the source code. We have selected a data-parallel model, based on nested parallelism, in which a program is expressed as a single sequence of operations, each involving multiple levels of parallelism. We are developing a high level programming language NESL, embodying the nested parallel model, along with compilation and run time support on several machine types. We are evaluating the model and language by implementing a number of basic algorithms and applications, and comparing the performance to implementations tailored to the specific machines.

5.1.1 First full release of NESL

We recently made available via anonymous ftp our first full release (Version 3.0) of our parallel language NESL. This release includes the implementation for workstations, the Connection Machines CM-2 and CM-5, and the Cray C90 (a version for the MasPar MP2 is also available from the University of North Carolina). The current implementation of NESL compiles into an intermediate language called VCODE which is linked with a machine-specific library of optimized parallel operations (CVL).

This is a complete and quite robust implementation of the NESL language, including:

- *Source code:*
 1. The NESL compiler and environment.
 2. The VCODE interpreter.
 3. Serial CVL.
 4. Cray CVL.
 5. CM-5 CVL.
 6. CM-2 CVL.
 7. An Xwindow graphics interface.

We have paid careful attention to code portability. The NESL compiler and environment have been tested under four different Common Lisps (Allegro, Lucid, CMU and Kyoto). The code has also been extensively tested across all the parallel machines we support. NESL has been used by over 100 users at several sites.

- *Documentation:*
 1. The NESL language definition (CMU-CS-93-129).
 2. The NESL user's manual (includes information on the profiler, tracer, remote execution, background processing and various environment variables).
 3. The VCODE reference manual (CMU-CS-91-118).
 4. The CVL reference manual (CMU-CS-93-114).

- *Working examples of NESL code:*

adaptive-integration

Adaptive integration of single-variable functions.

awerbuch-shiloach Algorithm for finding the connected components of a graph.

convex-hull The QuickHull algorithm for finding convex hulls. Includes a graphics demo.

hash-table An implementation of a parallel hash table.

line-fit Least-squares fit of a line to a set of points.

micro-shell A micro shell that keeps track of current directory and executes commands.

nas-cg The NAS conjugate gradient benchmark.

order Recursively finds the k^{th} largest element of a vector.

primes Work-efficient parallel implementation of the prime sieve algorithm.

<i>separator</i>	Geometric separator code. Includes a graphics demo of it running on an airfoil.
<i>sort</i>	Various sorts: quicksort, Batcher's bitonic sort and Batcher's odd-even mergesort.
<i>spectral</i>	Spectral separator code. Includes a graphics demo of it running on the same airfoil.
<i>string-search</i>	Fast string search algorithm.

These examples range from small to medium sized applications. They both serve as illustrations of how to write code in NESL, and also as useful utilities on their own.

The NESL front end runs on a user's workstation while allowing the remote execution of programs on parallel or vector supercomputers. The interactive environment includes a library of graphics routines, profiling and tracing facilities, and on-line documentation.

5.1.2 The current state of NESL

Given that NESL is a research language, an obvious question "is how practical is it to use?" In its current state we view NESL as a good language for the following uses:

- *Prototyping:* In NESL it is relatively easy to develop quick implementations of parallel algorithms and applications. As long as the algorithm itself is efficient, these implementations will usually run within an order of magnitude of machine specific code, and can often run within a factor of 2 (see the discussion below). NESL has been used, for example, to experiment with various parallel algorithms for the N-body problem, and various algorithms for finding separators of graphs for use in finite-element code.
- *Distributing lightweight portable code:* Although the full NESL environment is quite heavy, since it runs within Common Lisp, once an application is complete it is possible to dump the intermediate code (VCODE) to a file—NESL supplies utilities to do this. This code can then be executed directly using the VCODE interpreter: The interpreter binary only requires about 200 Kbytes space on workstations and a little more on the parallel machines. This makes it very easy to distribute applications: All the user needs is the VCODE interpreter and the dumped VCODE file. We have successfully used this technique for distributing geometric separator code.
- *Teaching parallel algorithms:* NESL's interactive environment, which runs on a local workstation while allowing remote execution, makes it very convenient for students to use in a class. The online help and graphical interface were also designed with teaching in mind. NESL has been used for teaching at several sites including Carnegie Mellon, Dartmouth, University of North Carolina, University of Toronto, and the University of Exeter (UK).
- *Compiler experimentation:* This was our major motivation in designing NESL. Because its core is small, simple, and designed from the start with parallelism in mind, it is well suited for this purpose. It allows one to work on compiler issues without having to deal with language artifacts that often arise in parallel extensions to serial languages.

Our implementation of NESL has concentrated as much on ease of use and true portability as on performance. Several implementation decisions reflect this concentration. For example, the use

of an interpreter has greatly simplified the portability and the ability to use the language interactively at the cost of performance (although the interpretation does not affect efficiency nearly as much as one might expect since the interpretive overhead is amortized over operations on large data sets).

For many applications it is possible to get within a factor of two of optimized machine specific code using NESL. However, there are a handful of things on which NESL does not perform particularly well, the most important being:

- NESL is not optimized for applications on dense matrices. In particular NESL does not support multidimensional arrays directly. Multidimensional arrays can be implemented with nested sequences, but this imposes some overheads—in particular, the arrays will not get laid out across processors in a communication-efficient way. We felt that HPF is making very good progress on computations on dense arrays and did not want to duplicate their work.
- Because the intermediate language VCODE is interpreted, the average parallelism has to be quite high to get good efficiency. For example, on a 32 processor CM-5 it is necessary to have over 1000-fold parallelism to get reasonable performance.
- The current CM-5 implementation is not optimized. In particular it does not use the vector units.
- Sequences in NESL are laid out across processors using a block distribution. To take advantage of locality the user has to understand this design. Furthermore, nested sequences are flattened so that each subsequence is likely to be mostly local. In general this a good heuristic for taking advantage of locality, but there are some cases where naive use of such a layout can cause memory contention at a single processor (e.g., several processor trying to access the same subsequence).

5.1.3 Parallel list ranking

List ranking and list scanning are two primitive operations used in many parallel algorithms that use list, tree, and graph data structures. But vectorizing and parallelizing list ranking is a challenge because it is communication-intensive and dynamic. In addition, the serial algorithm is simple and has small constants. In order to compete, a parallel algorithm must also be simple and have small constants. A parallel algorithm due to Wyllie is such an algorithm, but it is not work efficient—its performance degrades for longer and longer linked lists. In contrast, work efficient PRAM algorithms developed to date have very large constants.

We have developed a new, fully vectorized and parallelized algorithm that both is work efficient and has small constants. It does not achieve $O(\log n)$ running time, but we contend that work efficiency and small constants is more important, given that vector and multiprocessor machines are used for problems that are much larger than the number of processors and, therefore, the $O(\log n)$ time is never achieved in practice. In particular, to the best of our knowledge, our implementation of list ranking and list scanning on the Cray C90 is the fastest implementation to date. In addition it is the first implementation of which we are aware that outperforms fast workstations. The success of our algorithm is due to its relatively large grain size and simplicity of the inner loops, and the success of the implementation is due to pipelining reads and writes through vectorization to hide latency, minimizing load balancing by deriving equations for predicting and optimizing performance, and avoiding conditional tests except when load balancing.

5.1.4 Automatic parallelization of complex scan algorithms

We have implemented and published results on a technique to extract parallel reduction and prefix operations from serial loops. Our technique can extract such operations where existing techniques cannot. Furthermore, our technique is resilient to syntactic variations in the source code, whereas many existing techniques are sensitive to syntactic quality.

The compiler accepts sequential Fortran and for each recurrence it detects, tries to extract an equivalent parallel prefix operation. If successful, the compiler generates an SPMD program to execute the operation on the iWarp parallel computer.

We plan on extending the technique to work with a wider variety of recurrences. Possibilities include combining send, multiprefix, and pointer jumping operations.

5.1.5 Theoretical issues in parallel computing

In 1988 Leighton, Maggs, and Rao showed that for any set of packets whose paths have congestion c and dilation d , in any network, there exists a schedule for delivering the packets from their origins to their destinations in $O(c+d)$ steps while queuing at most a constant number of packets at any node at any time step. The proof, however, was non-constructive, and whether there existed a polynomial time algorithm for constructing the schedule remained an open problem for six years. We have resolved the question by demonstrating a nearly linear-time algorithm for constructing the schedule.

We have been studying simple online algorithms for processing connection requests in distributed networks, "call admission" algorithms. Each request comes with a source, a destination, and a bandwidth requirement. The duration of the request may or may not be known when the request is made. The call admission algorithm either schedules the request to begin immediately, schedules it to begin after some delay, or rejects it. We have analyzed the performance of the algorithms on simple networks such as linear arrays, trees, and networks with small separators. Three measures are used to quantify performance: makespan, maximum response time, and data-admission ratio. Our results include a proof that greedy algorithms are $\Theta(\log N)$ -competitive with respect to makespan on N -node trees, and a proof that no algorithm is better than $\Omega(\log N)$ -competitive with respect to data-admission ratio on a linear array, if each request can be delayed for at most some constant times its (known) duration.

We have been studying the behavior of a simple, local, load-balancing algorithm that redistributes unit-sized "jobs" among the nodes in a distributed network. At each step, each node sends a job to any neighbor holding fewer jobs. Previously, we showed that the algorithm balances the number of jobs at the nodes to within a difference of $O(d \log n / \mu)$ in $O(\Delta \log(n\Delta) / \mu)$ steps, where d is the maximum degree of any node in the graph, n is the number of nodes in the graph, Δ is the initial imbalance, and μ is the expansion of the graph. We have improved the bound on the running time to $O(\Delta / \mu + d \log n / \mu^2)$, which is optimal. In addition we show that reducing the imbalance below $O(d \log n / \mu)$ may require $n^{1-\epsilon}$ time, for any $\epsilon > 0$.

We have also shown that even if every node in an N -node butterfly network fails with some constant probability $p > 0$, with high probability it is still possible to identify some subset of at least $\Theta(N)$ nodes between which any permutation can be routed in $\Theta(\log N)$ steps. Although the analysis is quite involved, the routing algorithm itself is nearly as simple as the algorithm for

routing in a fault-free butterfly network. Previous algorithms for tolerating a large number of random faults in a butterfly network were extremely complicated and required more than $\Theta(\log N)$ time.

5.1.6 MPI implementation of CVL and NESL

We ported CVL to the new MPI (Message Passing Interface) standard, which is supported by both industry and academia. This guarantees the portability of CVL (and NESL) to future MPP machines based on a message-passing architecture and also to any other machines that support an efficient MPI implementation atop, for example, shared memory. We tested an initial version of MPI CVL on workstations (the Intel Paragon, IBM SP-1, and TMC CM-5), and released it for anonymous FTP by our users. Communication performance results on coarse-grained machines are within a few percent of the underlying message passing layer. Results on the fine-grained CM-5 are only 2/3 of those achieved by a machine-specific, CM-5 CVL, due mainly to the extra buffering needed to amortize the overhead of MPI's coarse-grained messages. We are working with the authors of the MPI reference implementation to correct some of these problems.

5.1.7 NESL implementation of the N-body problem

Using NESL we compared two algorithms for the three dimensional N -body problem, the Barnes-Hut algorithm and Greengard's Fast Multipole Method (FMM), to determine which of the two performs better in practice. Although FMM has a better asymptotic running time, $O(N)$ instead of $O(n \log N)$ for uniform distributions, the algorithm is significantly more complicated and it is not immediately clear above what values of N it performs better in practice. We studied the dependence of accuracy on the variable parameters Θ (in the Barnes-Hut algorithm) and p (in the FMM), and then compared the floating-point operation-count for the two algorithms for similar levels of accuracy. At a high level of accuracy (RMS-error $< 10^{-5}$), the FMM did fewer operations than the Barnes-Hut for $N > 10^4$, assuming a random distribution of points. At a lower level of accuracy (RMS-error $< 10^{-3}$) the FMM did not outperform Barnes-Hut until $N > 10^8$. We found that there was more parallelism than required and had to serialize some of the code to reduce the memory requirements.

5.1.8 NESL implementation of preconditioners

Solution of partial differential equations by either the finite element or the finite difference methods often requires the solution of large, sparse linear systems. When the coefficient matrices associated with these linear systems are symmetric and positive definite, the systems are often solved iteratively using the preconditioned conjugate gradient method. We have developed a new class of preconditioners, which we call support tree preconditioners, that are based on the connectivity of the graphs corresponding to the coefficient matrices of the linear systems. These new preconditioners have the advantage of being well-structured for parallel implementation, both in construction and in evaluation.

We evaluated the performance of support tree preconditioners by comparing them against two common types of preconditioners: those arising from diagonal scaling, and from the incomplete Cholesky decomposition. We solved linear systems corresponding to both regular and irregular meshes on the Cray C-90 using all three preconditioners and monitored the number of iterations required to converge and the total time taken by the iterative processes. We show empirically

that the convergence properties of support tree preconditioners are similar, and superior in many cases, to those of incomplete Cholesky preconditioners, which in turn are superior to those of diagonal scaling. Support tree preconditioners require less overall storage, less work per iteration, and yield better parallel performance than incomplete Cholesky preconditioners. In terms of total execution time, support tree preconditioners outperform both diagonal scaling and incomplete Cholesky preconditioners. Hence, support tree preconditioners provide a powerful, practical tool for the solution of large sparse systems of equations on vector and parallel machines.

5.1.9 Parallel computing and program development

Implementation of parallel algorithms in NESL

We have implemented NESL parallel versions of the following applications:

- Barnes Hut N-body solver.
- Greengard's fast multipole N-body solver.
- A hybrid N-body solver: the Parallel Multipole Tree Algorithm (PMTA).
- A fluid flow simulator (for non-viscous, incompressible fluids).
- Parallel algorithm for triangulating a set of points (Delaunay triangulation).
- Parallel graph separator (mesh partitioner).

The ability to bring up these applications quickly gives some evidence of the ease of use of the language. Visualizations of many of these algorithms are available at the Scandal web site, <http://www.cs.cmu.edu/~scandal>.

Memory management for parallel programming languages

We have studied scheduling techniques that save memory in implementing fine-grained parallel languages. The problem we are addressing is that these languages (including HPF, C*, Sisal, and NESL) can often create too much parallelism which in turn requires excessive memory. We developed a scheduling order for which we can prove that a parallel execution will never use significantly more memory than a sequential execution. The technique should be applicable to a wide class of languages.

Improved visualization tools for NESL

We have extended the NESL's visualization tools so that much more of the work gets executed in parallel on the machine that generates data. In particular all the coordinate transformations as well as line clipping now execute in parallel. This approach significantly improves the performance of realtime visualization, where the graphics can often become the bottleneck of an application.

5.1.10 Management of network resources

Near-optimal packet routing schedules

In 1988 Leighton, Maggs, and Rao showed that for any set of packets whose paths have congestion c and dilation d , in any network, there exists a schedule for delivering the packets from their origins to their destinations in $O(c+d)$ steps while queuing at most a constant number of packets at any node at any time step. The proof, however, was nonconstructive, and whether there existed a polynomial time algorithm for constructing the schedule remained an open problem. Leighton and Maggs resolved the question by demonstrating a nearly linear time algorithm for constructing the schedule [Leighton and Maggs 95].

Analysis of call admission algorithms

We have been studying algorithms for processing connection requests in distributed networks. These algorithms are called call admission algorithms. Each request comes with a source, a destination, and a bandwidth requirement. The duration of the request may or may not be known when the request is made. The call admission algorithm either schedules the request to begin immediately, schedules it to begin after some delay, or rejects it. In [Feldmann et al. 95] we analyze the performance of the algorithms on simple networks such as linear arrays, trees, meshes, and networks with small separators. Three measures are used to quantify performance: makespan, maximum response time, and data-admission ratio. Their results include a proof that greedy algorithms are $\Theta(\log N)$ -competitive with respect to makespan on N -node trees, and a proof that no algorithm is better than $\Omega(\log N)$ -competitive with respect to data-admission ratio on a linear array, if each request can be delayed for at most some constant times its (known) duration.

Analysis of network load balancing algorithms

We have been studying the behavior of a simple, local, load-balancing algorithm that redistributes unit-sized "jobs" among the nodes in a distributed network. At each step, each node sends a job to any neighbor holding fewer jobs. Previous work showed that the algorithm balances the number of jobs at the nodes to within a difference of $O(d \log n / \mu)$ in $O(\Delta \log(n \Delta) / \mu)$ steps, where d is the maximum degree of any node in the graph, n is the number of nodes in the graph, Δ is the initial imbalance, and μ is the expansion of the graph. Maggs, Richa, and several collaborators [Ghosh et al. 95] have now derived a tight analysis of the algorithm. They show that the actual time bound is $O(\Delta/\mu)$. In addition, they show that reducing the imbalance below $O(d \log n / \mu)$ may require $n^{(1-\epsilon)}$ time, for any $\epsilon > 0$.

Routing in butterfly networks with faulty components

Cole, Maggs, and Sitaraman have shown that even if every node in an N -node butterfly network fails with some constant probability $p > 0$, with high probability it is still possible to identify some subset of at least $\Theta(N)$ nodes between which any permutation can be routed in $O(\log N)$ steps. Although the analysis is quite involved, the routing algorithm itself is nearly as simple as the algorithm for routing in a fault-free butterfly network. Previous algorithms for tolerating a large number of random faults in a butterfly network were extremely complicated and required more than $O(\log N)$ time.

5.2 Formal hardware verification and symbolic manipulation

Our efforts in formal verification span multiple levels of abstraction, ranging from concurrent systems communicating by complex protocols at the highest level to transistor-level circuit implementations at the lowest. Currently we are focussing our efforts on three classes of systems: reactive systems, sequential processors, and arithmetic circuits. We believe that these different classes of systems call for different verification strategies in terms of how the desired behavior is specified and how system correctness should be established.

A *reactive system* consists of a number of independent agents communicating and synchronizing according to some protocol. Rather than describing the exact behavior of such a system, the specification consists of a set of general properties that the agents and the protocol must fulfill. For example a distributed shared memory system contains a number of bus controllers, memories, and caches interacting according to a cache coherency protocol. The specification states that the effects of read and write operations must satisfy some general consistency properties without fixing the exact system behavior. These systems have complex control but perform only simple data operations. We have recently begun focussing on the real-time behavior of such systems.

A *sequential processor* conceptually executes a single sequence of operations. The processor has a user-visible state, and the effect of each operation is to modify this state. For example a microprocessor executes a single sequence of instructions, each modifying the state of the CPU registers and the processor memory. Note that the underlying implementation may have complex temporal behavior due to pipelining, multiple functional units, and superscalar operation. The goal in verifying such a system is to show that, despite these complexities, the overall behavior corresponds to the single sequence view of the specification. Complexities in these systems stem from interactions between the control and data operations.

An *arithmetic circuit* implements some arithmetic function such as integer or floating point multiplication. Such systems have very simple control, e.g., combinational logic or a fixed execution sequence. On the other hand, their manipulation of data can be quite complex. In addition the user should be able to give an abstract specification in terms of the data encodings (e.g., two's complement) and the arithmetic operation performed.

In the process of developing formal hardware verification programs, we have developed efficient packages for symbolic Boolean manipulation based on BDDs. Over the past few years, BDDs have been used for a wide range of tasks in digital system design and beyond. We continue research on BDDs as well as on generalizations beyond Boolean functions. Our most recent extension is to Binary *Moment* Diagrams (BMDs), see section 5.2.18. These data structure provide an efficient method for representing word-level arithmetic functions.

5.2.1 Extending SMV to use dynamic variable ordering

The SMV model checking system uses BDDs to represent transition relations and sets of states. This representation can often avoid the state explosion problem that occurs when complex circuits and protocols are analyzed. The size of the BDDs is very sensitive to the relative order of the state variables. With earlier versions of SMV, the user had to construct the variable ordering manually in order to control the size of the BDDs. This process was very time consuming and required a deep understanding of the behavior of BDDs. Usually, the ordering that was generated was not very good and thus increased the cost of the verification.

We have improved SMV by implementing the automatic variable ordering algorithm of Rudell. The basic idea is very simple. When the algorithm is activated, it selects one variable and tries to order it so that the total number of BDD nodes is minimized. This process is repeated for all variables. Like other hill-climbing algorithms for NP-hard problems, this algorithm only gives a local optimum. However, when this algorithm is used properly, the result can be very good. As an SMV program evolves during the design cycle and becomes more complicated, the BDD size may grow dramatically. The dynamic variable ordering algorithm can reduce the BDD size by very large factors each time it is used. Even without a satisfactory initial variable ordering, the extended SMV system often obtains a variable ordering that is much better than the ordering constructed by the user. This feature makes SMV much easier to use, since it is possible to write SMV programs for large systems without worrying too much about the behavior of the BDDs with respect to variable ordering.

5.2.2 Parameterized circuits

The most significant limiting factor in the verification of large circuits is the state explosion problem. The number of states for a circuit or finite state machine grows exponentially in the number of state variables. Thus, the number of states required to represent a circuit can quickly become prohibitively large. Symbolic model checking allows us to avoid representing states explicitly. Unfortunately, there are still circuits for which this is not enough. There are many examples of very large circuits which cannot be verified in the straightforward way. Once such type of circuit is called a parameterized circuit. These circuits consist of a number of very similar or even identical components. While the number of components for any instance of a parameterized circuit is finite, the number can be arbitrarily large. Our hope is that we can exploit the duplication of components in these circuits to not only reduce the size of the verification task, but also to be able to verify an entire class of circuits simultaneously by verifying the circuit independent of the number of duplicated components.

The first attempt to address this question resulted in *indexed CTL*. By establishing an appropriate relationship between a circuit with n components and a circuit with $n+1$ components, one can guarantee that all instances of the parameterized circuit satisfy the same formulas in the indexed logic. The next step is to exhibit a bisimulation between an n component system and for example a 2 component system. Then by applying model checking to the 2 component system, we can also verify the n component system. However, this approach requires the bisimulation to be found by hand since finite state methods cannot be used to check a system with an arbitrary number of states. A method proposed by Kurshan and McMillan and independently by Wolper and Lovinfosse avoids this problem. This method uses a process q as an invariant when verifying a circuit composed of duplicate processes p . If we can show that a process p satisfies the invariant q and that $p \parallel q$ also satisfies q then we can use induction to show that any composition $p \parallel \dots \parallel p$ also satisfies q , independent of the number of duplicated components p . In addition, if q satisfies some specification we want to verify, then $p \parallel \dots \parallel p$ also satisfies this specification. Furthermore, this "satisfies" relation need not be as restrictive as the bisimulation required in the case of indexed CTL. A related approach uses graph grammars to generalize the "inherently" one dimensional composition used above to two dimensions. By first proving the equivalence of all networks derived from a grammar, one can reduce the task of verifying all systems derived from the grammar to verifying a representative. We are studying the possibility of extending the "satisfies" pre-order relation described above to graph grammars.

We are also currently exploring one promising approach for automatically generating the induction invariant. The idea is similar to the fixpoint computation technique employed in CTL model checking. We start with an initial *guess* invariant process q_1 which approximates the true invariant q , and we iteratively improve the guess invariant until no further improvement is possible or necessary. In particular, we start with $q_1 = p$. Clearly, q_1 is a proper representation of the process p . In general, q_i is an invariant process which properly represents the parallel composition of up to and including i copies of processes p . The guess invariant q_{i+1} can be obtained from q_i as follows: If the guess invariant q_i properly simulates the parallel composition of q_i and p , then $q_{i+1} = q_i$. Otherwise, we can obtain a counter-example c_i such that the parallel composition $q_i \parallel c_i$ can simulate the parallel composition of $q_i \parallel p$, and hence the parallel composition of up to $i+1$ processes of p (because q_i simulates up to i processes). Therefore, we can set $q_{i+1} = q_i \parallel c_i$. This process is repeated until we have reached a *fixpoint* for the invariant. That is, until $q_i = q_{i+1}$. The first q_i such that $q_i = q_{i+1}$ will be the invariant that we are looking for.

5.2.3 Formal verification of microprocessors

We have applied our methodology for verifying sequential processors to several actual circuit designs. In our methodology, the user describes the desired abstract system behavior as a series of assertions, each expressing the effect a given operation will have on some part of the system state. For a microprocessor, each assertion describes the effect of a single instruction execution on some part of the programmer-visible system state. The user then gives an *implementation mapping* describing how the actual circuit implements the abstract state. This mapping includes both spatial and temporal information, including cases where the state moves through various pipeline registers.

Our initial test for the methodology was Hector, a 16-bit nMOS microprocessor designed at North Carolina State University. We successfully verified a representative sample of instruction types and addressing modes. This is one of the first examples of formal verification being applied to an existing microprocessor circuit design, as opposed to one created specifically for formal verification.

More recently, we have begun working with the MIPS-X design from Stanford. This design is representative of modern, pipelined RISC processors. Our main difficulties to date have been in dealing with switch-level circuit modeling problems. This design appears to be well-suited for our verification methodology. The uniformity of the pipeline structure makes for a relatively simple state mapping despite the heavy pipelining of instructions. The clean interface with the memory subsystem (including caches) means that we can separately verify the processor and the memory.

5.2.4 Image representation and analysis with binary decision diagrams

In an effort to explore wider applications of Binary Decision Diagrams (BDDs) we have looked at representing bit-mapped images as Boolean functions from a set of variables giving binary representations of the X and Y coordinates to binary values indicating black or white. The BDD structure would then naturally exploit the hierarchy and uniformity in the image due to a sharing of common subgraphs.

One promising application of this approach is for representing VLSI masks. Highly structured designs, such as memory and array structures should have compact representations as BDDs. We have formulated the various spacing and width checks of design rule checking in terms of BDD operations. These operations exploit the sharing in the BDDs to give greater efficiency.

We have implemented tools for converting chip layouts (in Magic format) into BDDs and for performing mask checking operations. Our initial results have been respectable, but not spectacular, in terms of the sizes of the representations and the efficiency of the checking operations.

5.2.5 Inductive Boolean function manipulation

We have been working on a methodology for verification of parametric circuits based on symbolic manipulation of inductive Boolean functions. This methodology can be used to perform automatic proofs by induction both in the structural domain for parametric combinational circuits, and in the temporal domain for finite state sequential systems. It comprises:

- IBF schemata—which include a canonical representation for functions in a particular class of inductive functions and the symbolic manipulation algorithms,
- circuit representation mechanisms—in order to handle practical examples of parametric circuits, and
- appropriate manipulations to perform specific verification tasks.

We now have a practical implementation of both the core package for handling one class of inductive Boolean functions called Linearly Inductive Functions (LIFs), and the circuit representation mechanisms. We have experimented with this package and applied it to the task of checking correctness properties for some common combinational circuits such as an adder, a decoder etc. We have also applied it to obtain results for MCNC sequential benchmarks—both for obtaining canonical representations for circuit outputs, and for checking input/output equivalence of two finite state machine descriptions. Some more work needs to be done on looking into variable orderings, possibly extending it to use dynamic variable reordering.

Along the theoretical side, we have explored further the relationship between a canonical LIF representation of a sequential function output, and a classic DFA representation. We can show that an LIF representation of a regular language is isomorphic to a minimal DFA representation of the reverse language, i.e. one accepting the reverse strings. In principle a reverse DFA representation can have exponentially fewer (or more) states than a forward DFA representation. In practice, structured datapath circuits like shift registers, stacks, fifos, register files exhibit an exponential collapse. On the other hand, typical controller circuits exhibit an increase. We are working on identifying additional characteristics for classes of circuits with small reverse DFAs.

5.2.6 A model checker for a VHDL subset

The VHDL hardware description language is widely used in digital circuit design and is likely to be used even more in the future. Ensuring the correctness of descriptions written in VHDL is therefore very important. One of the most powerful techniques for formal verification of finite-state systems is temporal logic model checking. We have identified a subset of VHDL that accommodates the most commonly-used description styles and allows efficient verification using model checking techniques. Using this subset, we plan to build a model checker that can handle industrial designs of realistic complexity.

In order for the model-checking techniques to be accepted in industry, we believe that it is essential to provide an interface between the verification tools that we have developed and some widely-used hardware description language. VHDL is the obvious choice for such a language: It is used as the input language for many CAD systems, it provides a wide variety of descriptive styles, and it is an IEEE standard. However, VHDL is a complex language, and some of its features cannot be handled by model checking techniques. For verification of hardware designs it is therefore necessary to define a subset of VHDL that has a well-defined formal semantics and allows efficient translation and verification.

VHDL is a very rich language. Certain language constructs can be expressed in terms of more basic ones, while maintaining the same semantics. Therefore, for the first prototype, we have attempted to select the subset of language constructs which are most widely used, without restricting significantly the set of behaviors that can be described.

The language subset provides *entity declarations* and *architecture bodies* as design units. The entity declaration is restricted to specifying a header with the interface ports; *generics* are not allowed. The subset does not accept *configuration declarations*. The binding of component instances to design entities has to be done by *configuration specifications*. Package declarations and package bodies are supported.

All VHDL sequential statements are supported, as are most of the concurrent statements (block, process, concurrent procedure call, concurrent signal assignment and component instantiation statements). By including component instantiation into the subset, descriptions that combine behavioral and structural description styles can be verified. Recursive subprograms and operator or subprogram overloading are not supported.

Only one process is allowed to assign on each signal. As a consequence, resolution functions and the signal types *register* and *bus* are not supported. In interface declarations, only modes *in*, *out* and *inout* are allowed. The subset does not allow the specification of *time expressions*, either in the wait statement (*timeout clause*) or in the signal assignment statement (*after clause*).

Since the verification technique that we use is based on searching finite-state models, only discrete and finite data types and data structures are allowed. Real, access, and file types are not supported.

In order to speed our implementation of the first working prototype, we will attempt to implement in the first stage a more restricted language subset. The first prototype will handle synchronous descriptions, for which all wait statements are on the clock edge, and which don't contain subprograms or loops with dynamic iteration count. In later stages of the project we will augment the model checker to lift these restrictions and handle additional language features.

5.2.7 Symbolic linear-time temporal logic model checking

The past thirteen years has seen considerable research on efficient model checking algorithms for branching-time temporal logics, in particular, Computation Tree Logic (CTL). Verification tools based on these algorithms have discovered nontrivial design errors in sequential circuits and protocols and are now beginning to be used in industry. On the other hand descriptions in Linear-time Temporal Logics (LTL) are often simpler than branching-time temporal logics, because

LTL does not require path quantifiers. Moreover LTL can describe some properties that cannot be described in CTL. Relatively little research, however, has addressed efficient model checking algorithms for linear-temporal logic, and practical verification tools are virtually non-existent.

We have shown that LTL model checking can be reduced to CTL model checking with fairness constraints. Using this reduction we have constructed a *symbolic* LTL model checker that appears to be quite efficient in practice. In particular we showed how McMillan's SMV model checking system can be extended to permit LTL specifications. We have developed a translator T that takes an LTL formula f and constructs an SMV program $T(f)$ to build the tableau for f . The tableau construction that we use is similar to the one described in our previous work. To check that f holds for some SMV program M , we combine the text of $T = T(\neg f)$ with the text of M to obtain a new SMV program $P = P(T, M)$. We add CTL fairness constraints to P in order to make sure that eventualities of the form $a \cup b$ are actually fulfilled (i.e. to eliminate those paths along which $a \cup b$ and a hold continuously, but b never holds). By checking an appropriate CTL formula on P we can find the set V_f of all of those states s such that f holds along every path that begins at s . The projection of V_f to the state variables of M gives the set of states where the formula f holds.

Note that our approach makes it unnecessary to modify SMV (or even understand how SMV is actually implemented). We have evaluated the approach on several standard SMV programs, including Martin's distributed mutual exclusion circuit and the synchronous arbiter described in McMillan's thesis. To ensure that the experiments were unbiased, we deliberately chose specifications that could be expressed in both CTL and LTL. The results that we obtained were quite surprising: For the examples we considered, the LTL model checker required at most twice as much time and space as the CTL model checker. Although additional examples still need to be tried, it appears that efficient LTL model checking is possible when the specifications are not excessively complicated.

Many other problems, such as testing inclusion and equivalence between various types of ω -automata, can also be reduced to CTL model checking. We plan to use the same basic approach to extend SMV for testing inclusion between various types of ω -automata. Moreover, in many of the applications of model checking to verification, it is important to be able to assert the existence of a path that satisfies some property. For example, *absence of deadlock* might be expressed by the CTL formula $AG\ EF\ start$ (regardless of what state the program enters, there exists a computation leading back to the *start* state). Neither this formula nor its negation can be expressed in LTL, so LTL model checking techniques cannot be used to decide whether the formula is true or not. Ideally, it should be possible to reason about linear-time and branching-time properties in the same logic (say, CTL^*). We believe this goal can potentially be realized. Emerson and Lei have shown how to reduce CTL^* model checking to LTL model checking. If the transformation outlined in this paper can be extended to incorporate their reduction, then it should be possible to develop a model checker that can handle both types of properties. The same basic approach can be used to extend SMV for testing inclusion between various types of ω -automata.

5.2.8 Timing issues in circuit verification

The ability to verify bounded properties is extremely important in the verification of synchronous circuits and systems in general. Unbounded properties give information about the correctness of the system, while bounded properties provide information about its performance. Knowing how long it will take for a circuit to respond can sometimes be as important as knowing that it will respond at all. In a complex circuit an abnormally slow component may compromise the behavior of the whole system. Previous research extended CTL to allow the expression of bounded properties. SMV was also extended to handle these properties.

Current research involves finding ways of allowing for more realistic models. SMV has a restrictive notion of time. All events happen in one time unit. This condition is not true of actual circuits, where events take differing amounts of time to occur. We propose another extension of SMV to overcome this restriction. In the extended model, a transition takes time n to occur, where n lies within a specified time interval. This feature allows for the implementation of non-unitary transitions. Non-determinism is also implemented by this feature, since n can lie anywhere within the given range, bringing the model closer to reality and permitting a more accurate verification of circuits.

This idea leads to a more general concept that distinguishes the passage of time and transitions. It is possible to have zero time transitions and time passing without state changes. Circuits often change state in negligible time and also often spend time in one. More elegant and powerful models can be constructed using this more general concept of time.

Other research involving timing issues in verification relate more to realtime systems than to circuits. Parallel composition of processes is a topic that deserves close attention. General algorithms are used for composing processes. SMV allows for synchronous composition, where all processes execute simultaneously, and interleaving composition, where any process, but only one, can transition at any point. However, in many cases this condition is not always true. In realtime systems, for example, it is very common to have one processor that executes all processes, alternating among them. But in this case the choice of which process to execute next is not nondeterministic, it relies on priorities and scheduling policies. A general composition algorithm does not necessarily obey the right policy. In this case the semantics of the composed model may be incorrect, or the scheduling policy may have to be explicitly implemented. A high overhead is then imposed on the model, and state explosion problems can easily arise.

5.2.9 Specification using timing diagrams

While temporal logic has been a compact and natural specification technique for many classes of system behaviors, it may be difficult for engineers to write specifications for certain types of systems directly using temporal logic. This happens frequently when the specification involves the description of sequences of ordered events, a situation common in many asynchronous, distributed systems. On the other hand timing diagrams are natural and intuitive ways to describe temporal behavior of digital circuits and are widely used in the industry. It would therefore be desirable to incorporate timing diagrams as an alternative to temporal logic specification in formally specifying the behavior of a circuit for verification purposes.

We are currently developing a methodology to translate timing diagrams systematically into temporal logic formulas or automata, which can then be directly used to verify the system using our

model checker. We have developed a proper formalism of timing diagrams tailored for specification purpose, that allows the expression of conditional constraints among transitions as well as sequential orderings of events. We are currently building a prototype graphical timing diagram editor that engineers can use to describe visually the temporal behavior of a system. Subsequently, the timing information captured will be translated to some equivalent temporal logic or automata specification for verification by the model checker.

When the system fails to meet the specification, our current implementation of the model checker will simply print out a path in the state transition graph that corresponds to a counter-example of system behavior that violates the specification. Because the textural representation of such state transition is not always intuitive, engineers may need to spend considerable time tracking down the source of error from the counter-example. From this point of view a timing diagram can potentially become an invaluable tool for revealing subtle design errors. We extract state assignments of different signals from the textural representation of the state transition path, and visually display them as waveforms. If a timing diagram were used to specify the system behavior, we could further display the constraints that are satisfied by the counter-example and indicate the constraints violated by such—which violations correspond directly to the error detected). We plan to further explore this direction as we proceed in the implementation of the timing diagram interface.

5.2.10 Language inclusion for ω -automata:

ω -Automata are finite state machines accepting infinite strings. Like a conventional automaton, an ω -automaton consists of a set of states, an input alphabet, a transition relation, and a start state. However, the notion of a final state does not make sense for a machine accepting infinite strings, so a different acceptance condition must be used. In fact the several types of ω -automata differ precisely in their acceptance conditions. This discussion concentrates on one specific type of ω -automata, Buchi automata. A Buchi automaton can be thought of as the “natural” extension of finite state automata to strings of infinite length. The acceptance condition for a Buchi automaton consists of a subset F of the set of states S of the automata. A Buchi automaton accepts a string if there is at least one state of F that is entered infinitely often during the computation.

ω -Automata are of interest in part because of their use in verification. There are two basic approaches to verification. One is the model checking approach exemplified by SMV. In this approach a process or circuit is modelled as a finite state machine and specifications are given as logical formulas that are tested to see if they are true of the finite state machine. The other approach, taken by COSPAN, rests on the idea of language inclusion between two ω -automata. Both the specification and the the implementation are modelled as finite state machines, and a check is made to see if the language of the implementation machine is contained in the language of the specification machine. We have extended SMV to allow language containment between ω -automata to be verified. A translator has been built to convert an instance of the language inclusion problem, given as a description of a pair of Buchi automata, into an instance of model checking, given as an SMV program. If SMV model checking results in a specification being false, the counter-example feature of SMV will then give a string that is accepted by the implementation machine and not by the specification machine. The counter-example mechanism of SMV will also give a state trace for each machine.

In our experience it has proven difficult to describe circuits as ω -automata, so we are currently investigating the possibility of testing language inclusion between a Kripke structure and an ω -automaton. This would allow us to continue to describe circuit models as Kripke structures while allowing us to provide an ω -automaton as a specification. In many cases, it may prove easier and more intuitive to reason about specifications using ω -automata than CTL.

5.2.11 Formal verification of sequential processors

In our methodology for verifying sequential processors, the user describes the desired abstract system behavior as a series of assertions, each expressing the effect a given operation will have on some part of the system state. For the case of a microprocessor, each assertion describes the effect of a single instruction execution on some part of the programmer-visible system state. The user then gives an *implementation mapping* describing how the actual circuit implements the abstract state. This mapping includes both spatial and temporal information, including cases where the state moves through various pipeline registers.

We have begun working with the MIPS-X design from Stanford. This design is representative of modern, pipelined RISC processors. This design appears to be well-suited for our verification methodology. The uniformity of the pipeline structure makes for a relatively simple state mapping, despite the heavy pipelining of instructions. The clean interface with the memory subsystem (including caches) means that we can separately verify processor and memory.

Our initial focus has been on the MIPS-X register file, which includes the bypass logic used to implement operand-forwarding in the pipeline. The high-level specification for an idealized (non-pipelined) register file—besides declaring the inputs, outputs, and internal state of the abstract model—contains *assertions* describing how this state would be updated for each cycle of register operation. In this somewhat simplified example we assume the file has a single read and a single write port (the MIPS-X file has two read ports). Each cycle involves first a write or no-op operation, followed by a read. The different assertions describe what values would be produced on the read port as well as how the register elements would be updated. Several cases are needed to cover the various possibilities of matching read and write addresses, different read and write addresses, etc.

Under normal operation, the MIPS-X register file operates as a simple memory, with the pipelining fully hidden. That is, a read will always return the most-recently written value, even though there is a two-cycle latency for the actual register write. To verify normal operation, we would therefore use a high level specification similar to that of the example, but extended to two read ports. The implementation mapping would describe how the buffering in the bypass logic creates the illusion of an unpipelined behavior.

To implement speculative execution through branches, MIPS-X implements a “squash” operation in the register file to cancel any writes from the previous two cycles. To verify this operation we must extend the high level specification to express this capability. We do this by declaring three copies of the register file state: “new,” “middle,” and “old.” During normal operation, writes update the new state, while the previous new and middle states are copied to the middle and old states. Reads always retrieve the new state. A squash operation has the effect of rolling back the state so that the old state is copied to the new and middle states. This example illustrates how we would handle cases where the pipeline structure of a system is partially

visible. Rather than fully exposing the structure, we attempt to create an abstraction that hides as much structure as we can. In this example we use an abstraction based on rollback—a natural model for speculative execution.

We have verified the normal read/write behavior of the register file. Our main difficulties to date have been in dealing with switch-level circuit modeling problems. We expect to complete the verification of the squash operation shortly and then proceed to a verification of the full data path.

5.2.12 Inductive Boolean function manipulation

We have developed a methodology for formal verification of inductively-defined hardware, based on symbolic manipulation of classes of inductive Boolean functions. One of these classes, called the linearly inductive functions LIFs, is useful for capturing both the structural induction in linear iterative arrays and the temporal induction in finite state sequential systems. Our previous work focussed on handling each of these domains individually—functional verification of parametric combinational circuits (regular in structure), and behavioral verification of sequential circuits (regular in time).

Our latest effort has been in trying to handle both structural and temporal domains simultaneously. We are currently exploring the following two approaches:

- *Use of LIF multiple parameter framework*— We use a generalization of our single parameter induction framework to handle multiple parameters. Essentially, an induction trajectory in a multiple parameter space is represented by a parameter decision tree. At present we follow certain restrictions on these trajectories in order to use simple BDD-like reductions to obtain canonical tree representations. These restrictions may limit the kind of circuits that can be represented as LIFs with both circuit size and time as independent parameters of description.

A particular technique we have found useful in this context is the introduction of a dummy parameter to capture induction along an opposite direction to a main parameter. This has allowed us to represent the sequential description of a shift register (also stack, FIFO) with arbitrary depth as an LIF. We are working on generalizing this trick to other circuits with bidirectional behavior.

- *Use of symbolic simulation with LIFs* —For circuits where we cannot capture both space and time within the LIF framework we are looking into an alternative technique for verification. Starting with LIF descriptions of circuits parametric in size, e.g. an i -bit adder, we use symbolic simulation for a fixed number of cycles. This approach gives us a handle on the circuit behavior, for an arbitrary circuit size, after say $1, 2, \dots, n$ cycles of simulation. In our i -bit adder example this can be used to derive the sequential behavior of an arbitrary-sized accumulator.

5.2.13 Design rule checking with BDDs

As part of our research in symbolic Boolean manipulation, we have begun investigating new applications of Binary Decision Diagrams (BDDs) and related data structures. BDDs have proved successful in representing the logic functions for digital circuits for a variety of CAD tasks. It is therefore natural to see what other applications could profit from BDDs.

We have implemented basic mask checking operations using algorithms that operate directly on the BDD data structures. These algorithms can exploit the sharing in the BDD structure to avoid repeated checking of identical geometric structures. Unfortunately, our initial experiments indicate that our algorithms are not competitive with existing mask checking algorithms, such as those implemented by Magic. The degree of sharing in the BDDs and the ability of our algorithms to exploit it are insufficient to make up for the time and space requirements of BDD manipulation. We plan to study more sophisticated ways of mapping layouts to BDDs so as to increase the degree of sharing.

5.2.14 Quantitative characteristics of real-time systems

A number of algorithms have recently been proposed for verifying the behavior of finite-state, real-time systems. These algorithms assume that timing constraints are given explicitly in some notation like temporal logic. Typically, the designer provides a constraint on response time for some operation, and the verifier automatically determines if it is satisfied or not. Unfortunately, these techniques do not provide any information about how much a system deviates from its expected performance, although this information can be extremely useful in fine-tuning system behavior.

We propose algorithms to compute quantitative timing information, such as exact upper and lower bounds on the time between a request and the corresponding response. Our algorithms provide insight into *how well* a system works, rather than just determining whether it works at all. They enable a designer to determine the timing characteristics of a complex system given the timing parameters of its components. This information is especially useful in the early phases of system design, when it can be used to establish how changes in a parameter affect the global behavior of the system.

We model a real-time system as a labeled state-transition graph, where each path corresponds to an execution trace of the actual system. This graph is implemented internally using binary decision diagrams (BDDs), which generally produce a very compact representation. By employing symbolic model-checking techniques, we are able to handle extremely large state spaces with up to 10^{30} states efficiently. We show how to determine the minimum and maximum length of all paths leading from a set of starting states (representing the request) to a set of final states (representing the response). We also present algorithms that calculate the minimum and the maximum number of times a specified condition can hold on a path from a set of starting states to a set of final states. These algorithms are also extended to *timed transition graphs* (TTG), a model in which transitions take more than one time unit to occur. We believe that the techniques developed can be adapted to other models of computation as well.

Other approaches for analyzing real-time system exist. For example, the *rate monotonic* scheduling theory (RMS) defines a priority assignment algorithm that guarantees optimal response time. The RMS theory proposes a schedulability test based on total CPU utilization; a set of processes (which have priorities assigned according to RMS) is schedulable if the total utilization is below a computed threshold. If the utilization is above this threshold, schedulability is not guaranteed. Moreover, this analysis only considers certain types of processes with limitations, for example, on periodicity and synchronization. Another approach to schedulability analysis uses algorithms for computing the set of reachable states of a finite-state system. The algorithms construct the model with the added constraint that whenever an exception occurs (e.g., a deadline is missed)

the system transitions to a special exception state. Verification consists of computing the set of reachable states and checking whether the exception state is in this set. No restrictions are imposed on the model in this approach, but the algorithm only checks if exceptions can occur or not.

We develop an analysis method that does not impose any restriction except that the system be modeled as a set of processes that run in parallel and are defined by state-transition graphs. For example, the actual functional behavior of each process can be modeled and analyzed. Schedulability is determined by computing the minimum and maximum execution times for all processes. The process set is schedulable if and only if each process is guaranteed to finish execution before its next period starts. Our technique always determines if the set of processes is schedulable or not, unlike RMS analysis, which may not provide any schedulability information if utilization is above the computed threshold. If the processes are not schedulable, our algorithms determine which specific deadlines are missed and by how much. When no deadline is missed, the same results provide response times for each process, an important performance measure for real-time systems.

To demonstrate how our tools work, we verify a simplified aircraft control system used in military airplanes. It is extremely important that time bounds are not violated in such systems. Because of the risks involved in the failure of an aircraft, only conservative approaches to design and implementation are routinely used. Many modern techniques for software design such as formal methods are not commonly employed. We believe that formal verification can be very useful in increasing the reliability of these systems by assisting in the validation of schedulability and response times of the various components.

The aircraft control system can be characterized by a set of sensors and actuators connected to a central controller. It analyzes sensor data and control the actuators. Our model describes this controller and determines whether its timing constraints are met. The requirements used are similar to those of existing military aircraft. The aircraft controller is divided into systems and subsystems, each of which performs a specific task in controlling the airplane:

- *Navigation*: Computes aircraft position.
- *Radar Control*: Receives and processes data from radars. It also identifies targets and target position.
- *Radar Warning Receiver*: *This system identifies possible threats to the aircraft.*
- *Weapon Control*: Aims and activates aircraft weapons.
- *Display*: Updates information on the pilot's screen.
- *Tracking*: Updates target position. Data from this system is used to aim the weapons.
- *Data Bus*: Provides communication between processor and external devices.

Timing constraints for each subsystem are derived from factors such as required accuracy, human response characteristics and hardware requirements. In order to enforce the different timing constraints of the processes, priority scheduling is used. The priority assignment has been done according to the rate monotonic scheduling theory.

We have implemented this control system in the SMV language. The SMV model checker has been used to verify its functional correctness, while its timing correctness has been checked

using the quantitative algorithms described. In order to optimize response time, we have implemented a pre-emptive scheduler. However, pre-emptability is a feature that may not always be available. Nonpre-emptive schedulers are easier to implement, and allow for simpler programs but usually increase response time for higher priority processes. To assess the effect of preemption in our system we have also implemented a non-preemptive scheduler.

Using the model described above, we were able to compute the schedulability of the system. This is one of the most important properties of a real-time system. It states that no process will miss its deadline. We were able to determine that the process set is schedulable using preemptive scheduling. From our results we could also identify many important parameters of the system. For example, most processes take less than half their required time to execute. This indicates that the system is still not close to saturation, although the total CPU utilization is high.

Moreover, our results showed that preemption does not have a big impact on response times in this example. Except for one process, all others maintain their schedulability if a non-preemptive scheduler is used. If a preemptive scheduler were expensive, reducing the CPU utilization slightly might make the complete system schedulable without changing the scheduler. By having such information, the designer can easily assess the impact of various alternatives to improve the performance, without having to change the implementation.

The computation of quantitative characteristics also provided other valuable results about the system being modeled, such as:

- The overhead associated with preemption by other processes. This information is extremely important for determining the amount of priority inversion in a system.
- How fast a subsystem responds to an event. For example, in this model, pressing the fire button generates a complex sequence of events before the weapons are actually fired. We were able to determine the overhead imposed by the firing protocol and how it affects the overall response time of the system.

The results computed by our algorithms provide hints about the real-time system behavior that can be useful in improving its performance. We have found this approach to be very flexible and we believe that our method can be extremely useful to designers during the development of real-time systems. We are confident that these techniques will prove practical in the verification of a variety of other realistic designs.

5.2.15 Counter-examples and witnesses in symbolic model checking

Complex state-transition systems occur frequently in the design of sequential circuits and protocols. During the past ten years, researchers at Carnegie Mellon have developed an alternative approach to verification called *temporal-logic model-checking*. In this approach specifications are expressed in a propositional temporal logic, and circuit designs and protocols are modeled as state-transition systems. An efficient search procedure is used to determine automatically if the specifications are satisfied by the transition systems.

One of the most important advantages of model checking over mechanical theorem provers or proof checkers for verification of circuits and protocols is its *counterexample facility*. Typically, the user provides a high level representation of the model and the specification to be checked. The model-checking algorithm either terminates with the answer *true*, indicating that the model

satisfies the specification, or gives a counter-example execution that shows why the formula is not satisfied. The counter-examples can be essential in finding subtle errors in complex designs.

The main disadvantage of model checking is the state explosion that can occur if the system being verified has many components that can make transitions in parallel. Recently, the size of the transition systems that can be verified by model checking techniques has increased dramatically after the introduction of ordered binary decision diagrams (OBDDs). By applying this technique verification of systems that have more than 10^{100} states has become possible. However, finding counter-examples is significantly more difficult when OBDDs are used in model checking instead of explicit state enumeration techniques, especially when fairness constraints are involved.

Although finding counter-examples is extremely important, as far as we know, there is no description of how to do this in the literature on model checking. We have developed an efficient algorithm to produce counter-examples and witnesses for model checking algorithms. The algorithm is, in fact, the one that is used in the SMV model checker developed at Carnegie Mellon and works quite well in practice. We show how the counter-example facility can be used to debug a subtle asynchronous circuit design. We also discuss how to extend our technique to more complicated temporal formulas. This extension makes it possible to find counter-examples for verification procedures based on showing language containment between various types of o-automata.

5.2.16 Extraction of state machines from transistor-level circuits

Many formal verification tools, including the SMV symbolic model checker, operate on a finite state model of the system where each transition corresponds to a complete clock cycle. On the other hand, low-level circuit models, e.g., at the gate or transistor level, utilize a fine-grained timing model. Typically, each state transition represents one level of gate delay. Multiple transitions are required for each phase of the clock cycle. An automatic tool for extracting a cycle-level model from a detailed circuit representation would bridge this gap. We have shown that our symbolic simulation tools, developed for detailed circuit verification, can be operated to generate a cycle-level circuit model. Given a circuit representation (as a transistor netlist), a description of the clocking, and a specification of the I/O signaling, our program can automatically identify the state elements in the circuit and generate their next state functions. Note that the user need not explicitly identify the circuit latches. The operation of the program proceeds as follows:

1. The transistor circuit is processed by the Tranalyze program to generate a logically-equivalent gate-level representation. The generated circuit faithfully captures such switch-level effects as dynamic charge storage and bidirectional signal flow.
2. All zero-delay cycles in the gate network are broken by inserting unit delays. Heuristics are used in an attempt to minimize the number of delays inserted.
3. Treating the delay element outputs as the maximal set of state variables, next-state functions are derived by symbolically simulating the circuit according to the specified timing and I/O signalling. The simulator uses BDDs to represent the symbolic functions.
4. The actual state variables are identified by starting with the output functions and iteratively adding the dependent variables.

5. The BDDs for all next state and output functions are translated into an SMV description of the cycle-level model.

We have successfully applied the tool to a number of circuits including static RAMs, stacks, and microprogram controllers. We have found the program quite successful at identifying a minimal number of state elements and at generating compact cycle-level models.

5.2.17 Verification of arithmetic circuits

Proving the correctness of arithmetic operations has always been an important problem. The importance of this problem has been recently underscored by the highly-publicized division error in the Pentium processor. Some people have estimated that this error cost Intel almost 500 million dollars. We have verified a division circuit that implements the floating point IEEE standard. The circuit uses a radix-four SRT division algorithm that is similar to the one used in the Pentium. The algorithm looks ahead to find the next quotient digit in parallel with the generation of next partial remainder. An 8-bit ALU estimates the next remainder's leading bits. A quotient digit look-up table generates the next quotient digit depending on the leading bits of the estimated remainder and the leading bits of the divisor.

We formalize the circuit and its correctness conditions as a set of algebraic relations over the rational numbers. These algebraic relations correspond closely to the bit-level structure of the circuit, and could have been generated mechanically from a hardware description. Most of the hardware for the SRT algorithm can be described by linear inequalities. We have proved the correctness of the circuit fully automatically using a powerful theorem prover called Analytica that we have developed. Analytica is the first theorem prover to use symbolic computation techniques in a major way. It is written in the Mathematica programming language and runs in the interactive environment provided by this system. Compared to Analytica, most theorem provers require significant user interaction. The main problem is the large amount of domain knowledge that is required for even the simplest proofs. Our theorem prover, on the other hand, is able to exploit the mathematical knowledge that is built into the symbolic computation system and is highly automatic.

5.2.18 Verification of arithmetic circuits with binary moment diagrams

We have developed a new method for verifying hardware implementations of arithmetic functions such as integer multiplication. Multipliers are not amenable to existing verification methods based on BDDs. The BDD representations of the Boolean functions for multiplication grow exponentially with the word size. This places a practical word size limit of around 16 bits on this approach, yielding data structures totalling 0.5 Gigabytes, with the size more than doubling with each additional bit.

The newly developed approach involves a hierarchical methodology. At the low level, basic building blocks such as adders, Booth encoders, and add-steppers are given at the bit level in terms of logic gate networks. These blocks are then abstracted to a word level, viewing each data word as encoding an integer (fixed point) or a rational (floating point) number. The word level functions for the blocks are then composed to determine the overall circuit function.

A key to the success of this approach has been the development of the Binary Moment Diagram

(BMD) data structure for representing word-level functions. BMDs are similar to BDDs, but have the property that they can represent functions such as multiplication efficiently.

To date a number of integer multiplier designs with word sizes up to 256 bits have been verified. The method can deal with the numerous circuit design techniques used, including carry-save addition, multiplier recoding, and sign bit correction.

Other promising applications of BMDs include symbolic methods for analyzing large Markov systems and symbolic manipulation of polynomial expressions.

5.2.19 Parallel BDD manipulation

Although the use of BDDs has greatly expanded the complexities of systems that can be automatically verified, the high memory requirements still limit their applicability. The most efficient BDD packages require in total around 22 bytes per BDD node, placing a limit of 5-10 million nodes before the physical memory space is exceeded. Beyond this, the performance becomes unacceptable due to thrashing of the virtual memory system.

Many researchers have attempted to map BDDs onto various parallel or distributed machines. Thus far, most of these efforts have been disappointing: speed-ups have been at best small and in some cases nonexistent. On the other hand, these implementations should be evaluated more on their ability to expand capacity rather than on their speed up. By this measure, modest, but not overwhelming, successes have been achieved.

We are investigating a new partitioning of BDD manipulation onto multiple processors. We believe this approach will greatly increase the capacity while potentially providing some speedup. In particular our implementation will map onto a linear array of processors, with each processor responsible for the nodes labeled by a contiguous subrange of the variables. The application program will issue requests to the processor at one end of the array. These requests will cause activity to propagate toward the other end, corresponding to the recursive invocations of BDD algorithms. Activity will then propagate back, corresponding to the returning recursive calls.

To exploit the parallel capacity of the array, the application must pipeline its requests and thereby hide the latency between when requests are sent to the array and when the response is received. We believe this can be done readily for most BDD applications and have begun implementing this approach using Dome, a parallel object library built atop PVM.

5.2.20 Reactive system verification

A reactive system consists of a number of independent agents communicating and synchronizing according to some protocol. Rather than describing the exact behavior of such a system, a specification consists of a set of general properties that the agents and the protocol must fulfill. For example, a distributed, shared-memory system contains a number of bus controllers, memories, and caches interacting according to a cache coherency protocol. The specification states (without fixing the exact system behavior) that the effects of read and write operations must satisfy some general consistency properties.

Bus performance analysis

Most verification algorithms assume that timing constraints are given explicitly in some notation like temporal logic. Typically, the designer provides a constraint on response time for some operation, and the verifier automatically determines if it is satisfied or not. Unfortunately, these techniques provide no information about how much a system deviates from its expected performance, although this information can be extremely useful in fine-tuning system behavior.

We have developed algorithms to compute quantitative timing information, such as exact upper and lower bounds on the time between a request and the corresponding response. Our algorithms provide insight into how well a system works, rather than just determining whether it works at all. They enable a designer to determine the timing characteristics of a complex, system given the timing parameters of its components. This information is especially useful in the early phases of system design, when it can be used to establish how changes in a parameter affect the global behavior of the system. These algorithms determine the minimum and maximum length of all paths leading from a set of starting states to a set of final states. We also present algorithms that calculate the minimum and the maximum number of times a specified condition can hold on a path from a set of starting states to a set of final states.

This method can produce several types of information. time to events is computed by making the set of starting states correspond to the event and the set of final states correspond to the response. Schedulability analysis can be done by computing the response time of each process in the system and comparing it to the process deadline. Performance can be determined in a similar way. Preliminary versions of the algorithms have been incorporated into the SMV model checking system and have been used to verify several nontrivial systems, including the PCI Local bus.

PCI is a high performance bus architecture designed to become an industry standard for current and future computer systems. It is used primarily in systems based on Intel Pentium or DEC Alpha processors. We have modelled the PCI bus, concentrating on its temporal characteristics, and analyzed its performance. We have computed transaction response time in various configurations of the system and we have been able to bound the response time of a PCI transaction as well as to produce detailed information about each phase of the communications protocol. In addition, we have computed the overhead imposed by arbitration, bus acquisition, and other phases of the protocol. This type of information allows the designers to understand the behavior of the system more accurately than the information generated by traditional verification methods. Our results also uncovered subtleties in the behavior of the system that could have been difficult to find otherwise.

Multilevel verification

We have developed a new methodology for formal verification of hardware designs by using a technique that combines the strengths of symbolic trajectory evaluation and symbolic model checking. In earlier work we developed a methodology for formally verifying data-intensive circuits. That methodology involves specifying a system by giving high-level assertions over abstract states and state mapping for the abstract states. The assertions define a set of transitions in an abstract Moore machine. We express these assertions in Hardware Specification Language (HSL). These assertions are quite abstract, and hide low level details such as detailed signal timing and pipelining. From the HSL assertions and the state mapping, simulation patterns are

generated for a symbolic trajectory evaluator and the circuit is verified as implementing the state machine defined by the assertions. This technique can efficiently verify systems that can be viewed as state transformation systems—systems, such as microprocessor data path and memory circuits, which have well-defined state. However, this methodology has a drawback stemming from its limited notion of time. Since a HSL assertion involves only the current state and its successor, many important temporal properties of a system, such as absence of deadlock, can't be easily expressed.

This limitation can be overcome by checking the state transition system defined by the HSL assertion against specifications in the temporal logic CTL. In our verification methodology we first verify that a switch-level or a gate-level circuit implements the state machine defined by the HSL assertions. Next we do CTL model checking on that machine. Assertions are abstract description of the systems that abstract away many low-level details of the circuits—considerably simplifying the finite state system which we must model-check, as compared to one derived directly from the circuit. Also, having an HSL assertion description makes it possible to incorporate automatically abstraction techniques to simplify the system that is model checked. Our methodology supports a modular design discipline in which a system composed of different units at different levels of abstraction can be verified. Currently we have a translator which renders an HSL specification into a description of a finite state system in SMV. We have verified temporal properties of many small circuits including FIFO's with nondeterministic controllers.

5.3 Formal verification of sequential processors

This component of the grant concentrates on formally verifying that a low-level circuit design implements a high-level specification for a class of systems known as "sequential processors." Systems in this class operate on an externally visible, stored state, following a single thread of control. Examples include both complete microprocessors, as well as subsystems such as memories, datapath, and floating-point units.

For verifying such systems, we employ a form of symbolic simulation known as "symbolic trajectory evaluation." The verifier effectively simulates the behavior of the circuit over a set of patterns encoding all possible state and input values, derived from a high-level specification of the intended system behavior. This approach can also operate with low-level system models capturing such effects as detailed timing and transistor-level behavior. We are applying this approach to the verification of pipelined microprocessors as well as to memory arrays.

Our verifiers use graph representations of discrete functions to encode the symbolic state representations. These include Binary Decision Diagrams (BDDs) for representing bit-level circuit operation, and Binary Moment Diagrams (BMDs) for representing the word-level behavior of arithmetic circuits.

5.3.1 Applications of symbolic trajectory evaluation

In [Bryant 95a] we show how symbolic trajectory evaluation could be usefully applied to circuits such as the floating point divider in the Intel Pentium(TM) microprocessor. It is impractical to verify multiplier or divider circuits entirely at the bit-level using BDDs, because the BDD representations for these functions grow exponentially with the word size. It is possible,

however, to analyze individual stages of these circuits using BDDs. Going beyond verification, we show that bit-level analysis can be used to generate a correct version of the table.

In [Pandey et al 96] we showed that symbolic trajectory evaluation can be used to verify memory arrays, consisting of storage with embedded logic. Examples of such arrays include random-access memories, multi-ported register files, caches, and content-addressable memories. This class of circuit is typically designed by hand at the transistor level, and hence the verification tool must be capable of dealing with low-level circuit models and very large state spaces (over 10,000 bits of state). Our verifier succeeds on this task, since it uses a switch-level simulation model and efficient encoding methods to represent the large state spaces.

5.3.2 Symbolic representations of discrete functions

In [Bryant 96] we describe work both at Carnegie Mellon and elsewhere on representing discrete functions with graph representations. This work builds on our pioneering work on Binary Decision Diagrams. Our recent work in this area involves the use of Binary Moment Diagrams (BMDs) to represent functions mapping Boolean variables to numeric values. Such functions can be used to express the behavior of arithmetic circuits operating on words of data. We are currently implementing an efficient BMD package that will improve performance by dynamically reordering the variables as well as dynamically changing function decompositions.

5.3.3 Automatic verification of sequential circuit designs

A model checker for a VHDL subset

In a previous stage of this project a core subset of VHDL (including the fundamental synchronization and communication paradigms) was defined. The next step has been the development of a VHDL front-end that produces an intermediate representation stored on disk and used by subsequent phases of the model checker. The prototype front-end is able to parse syntactically- and semantically-correct VHDL descriptions, and has been tested and demonstrated using a test suite.

We have also continued work on the elaboration algorithm, which automatically builds a symbolic, BDD-based model for a given VHDL architectural description. Models elaborated in this manner represent the behavior of the corresponding VHDL description, as defined in the language reference manual. A state of the model corresponds to the state reached by the VHDL description at the end of a simulation cycle. Hence, a transition of the model represents the effect of a VHDL simulation cycle. Therefore, it is possible to study the behavior of the VHDL description using symbolic simulation techniques on the generated model. In addition, this modeling approach can be used to prove temporal properties or check equivalence of VHDL models. A first version of this elaboration procedure has been implemented and demonstrated.

Performance verification of time-critical applications

We began studies on the formal aspects of Verus, the language used to describe time-critical circuits, and have completed a first draft of the semantics of the language. Work on the Verus compiler continued, using results of this first draft. Most of the aspects of the core language have already been defined. The implementation of the compiler already incorporates these aspects, and non-trivial programs can already be compiled.

We began research on selective quantitative analysis. This research originated from the observation that not all execution sequences of a model are equally interesting. In many cases it is more important to observe the behavior of the system under specific conditions. For example, one might be interested in determining the response time to an event provided that the rest of the system is not overloaded. This would produce the response time for an average situation, instead of the response time for the worst case. This result can provide important information about system behavior that might be difficult to obtain otherwise.

Selective quantitative analysis is one method to produce this type of information. It allows the user to specify a condition (as an LTL formula) that must be satisfied by the execution paths of interest. Using LTL model checking techniques, the model is then restricted to only those paths that satisfy the condition, and quantitative analysis is performed on the restricted model.

Reasoning about parameterized circuit design: induction

Traditionally, model checking has only been applied for verifying single finite-state systems. However, most hardware designs are parametrized. For example, the width of the integers is a parameter while designing a multiplier. These parameterized designs give rise to infinite family of finite-state systems. Our aim is to reason automatically about entire families of state-transition systems.

In our formalism, these families are described using context-free network grammars. We express state properties using regular languages. Our method finds an invariant using the structure of the grammar and the regular languages describing the state properties. This invariant is found by constructing an abstraction function based on the regular language and then using this abstraction function to derive the invariant.

We have verified two non-trivial families of circuits using our techniques. Our earlier idea was only applicable to synchronous models (Moore Machines). We have extended our techniques to handle asynchronous models as well. We have also developed an unfolding technique which will help us in deriving invariants. We plan to look at more examples.

Reasoning about parameterized circuit design: symmetry

Most large circuits are highly symmetric. For instance, it is possible to find symmetry in memories, caches, register files, bus and network protocols—any type of hardware containing replication of structures. It should be possible to exploit this symmetry in order to avoid searching the entire state space of the circuit or to reduce the size of the BDDs needed to represent the set of states. Our goal is to reduce the size of the state space explored during model checking by exploiting the inherent symmetry present in the system. Symmetries in systems are described by permutation groups acting on finite sets. The underlying symmetry group induces an equivalence relation on the set of states. Since the BDD size for this equivalence relation can be quite large, we must resort to certain implicit techniques so that we don't have to build the BDD for the equivalence relation. One such implicit method uses multiple representatives from each equivalence class.

We have developed techniques to do reachability with multiple representatives. Our objective is to develop techniques that enable us to do full CTL model checking with multiple representatives. We have applied our model checking techniques with multiple representatives to a simple cache coherence protocol based on the FutureBus+ IEEE protocol. The FutureBus+

protocol describes a scheme by which cache coherence can be maintained in a multiprocessor system. In the experiments we performed, we varied the number of processors and the number of cache lines. The savings in the BDD sizes by using symmetry were substantial. For example, in the configuration with 4 processors and 8 cache lines, we achieved a factor of 14 reduction in the BDD sizes.

We are currently implementing a model-checker, SYMM, which will support both explicit and symbolic state representations. SYMM will also allow the user to specify the symmetries of the system being verified. Using SYMM, we hope to compare explicit state representation (using symmetry) against symbolic state representation, which uses Binary Decision Diagrams (BDDs).

Compositional reasoning

Essentially, compositional reasoning allows one to reason about individual components of a system by making certain assumptions about a component's environment. As a result, we do not have to explore the entire state space of the system to verify the properties of interest. Instead, we determine properties of individual components and then use these properties to infer properties of the entire system. This approach, called "rely-guarantee" reasoning, has been used to verify synchronous circuits. We are currently investigating how to extend this paradigm to asynchronous circuits.

First, we continue to develop a more natural model of asynchronous computation that would allow for this type of reasoning. In our model, components have interleaved execution except that common actions can only be performed by synchronization. In other words, at any instant, either machine is allowed to perform an action (make a transition) unless that action also belongs to the other component's action set — in which case neither component may perform the action unless they perform it simultaneously (both machines make a transition).

We are also investigating how to compute a simulation relation between machines. Intuitively, M is related to N if every behavior or computation of M is a behavior or computation of N . This relation has the property that if M and N are related, then every property of N (every formula true in N) is a property of M (is true in M). Because we are concerned with asynchronous composition, two problems immediately stand out. First, we need to take into account fairness. We most probably want to disallow infinite computations in which a component only makes a finite number of steps. Second, because of the interleaving semantics, a component may remain in a state while the entire machine takes a step. If we are only interested in the behavior of the one component, then this step becomes what is called a stutter step (the machine takes a step, but the state does not change). Our simulation relation must therefore take into account the possibility of such stuttering.

Despite much added detail and complexity, this model has similarities with the synchronous case. In particular, this model has the same properties about composition and simulation that allowed for rely-guarantee style reasoning in the synchronous case. This would then provide a framework for compositional reasoning for both synchronous and asynchronous circuits.

5.3.4 Extensions of symbolic trajectory evaluation

In [Jain et al. 96] we describe a methodology and prototype tools for verifying nondeterministic implementations of systems with deterministic semantics. Such systems are found in many modern designs, where different units interact with one another via handshaking protocols, possibly stalling until operands or structural resources are available. Formal verification requires analyzing the system behavior under all possible timings of these interface signals. Our approach involves composing state machine descriptions of the individual signalling protocols to form a system control graph. We have extended our symbolic trajectory evaluator, a form of symbolic simulator, to verify all possible execution paths in the control graph. Symbolic variables are used to encode nondeterministic choice points, while fixed point algorithms are used to handle the self-loops representing stall conditions. This new approach has been applied to the fixed-point unit of a PowerPC implementation obtained from IBM. Verifying the correct processing of an OR instruction involves constructing a control graph of 75 state vertices and evaluating over 500 possible execution paths.

5.3.5 Hierarchical arithmetic circuit verification

In [Chen and Bryant 96] we describe an arithmetic circuit verifier ACV, in which circuits expressed in a hardware description language, also called ACV, are symbolically verified using Binary Decision Diagrams for Boolean functions and multiplicative Binary Moment Diagrams (*BMDs) for word-level functions. A circuit is described in ACV as a hierarchy of modules. Each module has a structural definition as an interconnection of logic gates and other modules. Modules may also have functional descriptions, declaring the numeric encodings of the inputs and outputs, as well as specifying their functionality in terms of arithmetic expressions. Verification then proceeds recursively, proving that each module in the hierarchy having a functional description, including the top-level one, realizes its specification. The language and the verifier contain additional enhancements for overcoming some of the difficulties in applying *BMD-based verification to circuits computing functions such as division and square root.

ACV has successfully verified a number of circuits, implementing such functions as multiplication, division, and square root, with word sizes up to 256 bits. BCD-to-Binary conversion circuits with word sizes up to 18 decimal digits have also been verified in reasonable time.

5.4 Parallel programming with NESL

This component of the grant concentrates on greatly simplifying the task of programming parallel machines. We released a new version (3.2) of our programming language NESL. We have made this release available via the WWW via the NESL home page

<http://www.cs.cmu.edu/~scandal/n esl.html>

A paper describing the language and the motivation behind it appears in [Blelloch 96]. We have written several papers that describe both the implementation ideas behind NESL and various applications that have been coded in NESL. The results of these papers are outlined below. We have also released a library of algorithms written in NESL, available off of the NESL home page, ranging from mesh partitioners to N-body codes. The PC++ group at Indiana has been looking at the nested parallelism ideas in NESL for use in their language. In our work we argue that NESL allows for much more concise and clean description of parallel applications than either HPF or MPI.

Our research is composed of two main components: (1) research in how to implement nested parallelism efficiently and (2) research on how various applications can use nested parallelism. Here we list recent results in both areas.

5.4.1 Implementing nested parallelism

In [Hardwick 96] we present work in progress on a new method of implementing irregular divide-and-conquer algorithms on distributed-memory multiprocessors. The algorithms are described at a very high level in a nested data-parallel language model. Then, using a mixture of compile-time and run-time techniques, they are efficiently translated onto code for distributed memory machines. The main features discussed are:

- Recursive subdivision of asynchronous processor groups to match the change from data-parallel to control-parallel behavior over the lifetime of an algorithm,
- Switching from parallel code to serial code when the group size is one (with the opportunity to use a more efficient serial algorithm),
- A simple, manager-based, run-time, load-balancing system.

Sample algorithms translated from the high-level, nested, data-parallel language NESL into C and MPI using this method are significantly faster than the current NESL system and show the potential for further speedup. The applications we consider are mesh partitioners, Delaunay triangulation, convex hull, and sorting.

In [Blleloch et al. 95a] we present a space-efficient scheduling algorithm for nested parallel languages. A notorious problem with the implementations of languages that supply fine-grained parallelism is that they can use much more memory than necessary. This has been observed in our implementation of NESL, as well as with implementations of ID, SISAL, Cilk, and Multilisp. Without careful scheduling, the parallel execution of a program on P processors can use a factor of P more memory than a sequential implementation of the same program.

We present a space-efficient algorithm for scheduling tasks for which we can place strong bounds on how much more memory the parallel implementation will require beyond a standard sequential implementation. The scheduling algorithm is a natural implementation based on maintaining a stack of tasks. The novel additions are a technique for delaying allocation of large blocks of memory, the way the stack is kept synchronized, and the proof that this combination guarantees our memory bounds. Based on these ideas, we present a provably good implementation of NESL [Blleloch and Greiner 96]. We have recently completed a preliminary implementation of these ideas on the SGI Power Challenge and have seen memory savings of a factor of three or better. We have applied for a patent for these ideas.

[Blleloch et al. 95b] considers issues of memory performance in shared memory multiprocessors that provide a high-bandwidth network and in which the memory banks are slower than the processors. We are concerned with the effects on performance of memory bank contention, memory bank delay, and the bank expansion factor (the ratio of number of banks to number of processors), particularly for irregular memory access patterns. This work was motivated by observed discrepancies between predicted and actual performance in a number of irregular algorithms implemented for the Cray C90 when the memory contention at a particular location is high. The work also applies to machines such as the Tera machine and to some extent to bus-based SMPs, such as the SGI Power Challenge.

We study how to implement "futures" efficiently in [Greiner and Blelloch 96]. Speculation using futures (as in Multilisp or Cool) or "leniency" (as in ID or Ph) is a common technique to expose parallelism in programs. Current implementations of these languages use queues for maintaining tasks that suspend waiting for a value of a variable (a future). When the value becomes available, the queue of tasks is reactivated. The problem with this approach is in how to implement the queue. Standard implementations use linked lists to maintain the queue. This can sequentialize code that appears parallel, since tasks have to be added and removed from the queue sequentially. Unfortunately, trees also do not seem to work, since it is costly to determine in parallel where in the tree a task should be added. In our work we have developed a data-structure based on dynamic arrays for efficiently implementing the queues in parallel (the arrays grow in powers of two as more tasks are added). We use amortized analysis to guarantee that the cost of maintaining the queues is slight.

5.4.2 Applications and algorithms

We describe in [Blelloch et al. 96] an efficient divide-and-conquer parallel algorithm developed and implemented in NESL. Delaunay triangulation is one of the most used methods for surface interpolation and the generation of unstructured finite-element meshes. Developing a practical parallel algorithm for Delaunay triangulation, however, has been notoriously hard because of the irregular and dynamic nature of the efficient algorithms. Although there have been many theoretical algorithms for the problem and some implementations based on bucketing that work well for uniform distributions, there has been little work on implementations for general distributions. In this work we have developed a divide-and-conquer parallel algorithm that does little more total work than the best sequential algorithm. Furthermore it quickly divides the problem into smaller problems which each can be run sequentially on separate processors. Our experiments use non-uniform distributions which appear in various scientific applications.

[Greiner and Blelloch 95] details work on comparing various algorithms for finding graph connectivity. Graph connectivity is an important subroutine in many applications, including computer vision (region labeling) and the Swendsen Wang algorithm for simulating Ising models. In this work we compare many different algorithms. The comparison not only looked at final running times, but also studied how various parameters such as communication, number of nodes processed, and number of edges processed were effected by the algorithm and the type of graph. All experiments were done using NESL.

We have developed a new class of preconditioners which we call support tree preconditioners ([Gremban et al. 95]). Solution of partial differential equations by either the finite element or the finite difference methods often requires the solution of large, sparse linear systems. When the coefficient matrices associated with these linear systems are symmetric and positive definite, the systems are often solved iteratively using the preconditioned conjugate gradient method.

We have developed a class of support tree preconditioners that are based on the connectivity of the graphs corresponding to the coefficient matrices of the linear systems. These new preconditioners have the advantage of being well-structured for parallel implementation, both in construction and in evaluation. We evaluated the performance of support tree preconditioners by comparing them against two common types of preconditioners: those arising from diagonal scaling and from the incomplete Cholesky decomposition. We did the original implementation in NESL and then generated an optimized version for multiple processors of the Cray C90. We show

empirically that the convergence properties of support tree preconditioners are similar, and superior in many cases, to those of incomplete Cholesky preconditioners, which in turn are superior to those of diagonal scaling.

Support tree preconditioners require less overall storage, less work per iteration, and yield better parallel performance than incomplete Cholesky preconditioners. In terms of total execution time, support tree preconditioners outperform both diagonal scaling and incomplete Cholesky preconditioners. Hence, support tree preconditioners provide a powerful, practical tool for the solution of large sparse systems of equations on vector and parallel machines.

In a special 50th anniversary issues of ACM Computing Surveys we were invited to contribute a paper that summarizes the current state-of-the-art in parallel algorithms [Blelloch and Maggs 96]. The paper will appear with many other papers that summarize other fields in computer science. It is a condensed version of the chapter on parallel algorithms that will appear in the *CRC Handbook on Computer Science*.

5.5 Automatic verification of sequential circuit designs

5.5.1 Automatic determination of time bounds for sequential circuits

We developed a method to use quantitative symbolic algorithms to analyze the behavior of a system. This method computes minimum and maximum delays between the occurrence of two events, as well as the number of times a specified condition occurs in such an interval. A more refined analysis is also possible by restricting the model to consider execution paths that only satisfy a certain condition. This can help in understanding how the system reacts to different conditions. To strengthen our verification methodology, we have extended the method to permit interval model checking, that is, checking a formula with respect to finite intervals. We have also extended the method by defining a new language, Verus, which has been specifically tailored to simplify the expression of real-time properties and constraints. Finally, we have begun integrating all these ideas into a tool for the analysis and verification of timing properties of sequential circuits. To demonstrate the efficiency of this method in the verification of complex systems, we have applied it to several real applications. The Verus compiler has had its first internal release. Nontrivial programs can be written, and interesting properties of such systems can be verified.

5.5.2 Reasoning about parameterized circuit designs

We have developed a technique to verify properties about infinite families of finite-state systems. For example, we want to reason that a bus protocol works correctly regardless of the number of processes connected on the bus. The general technique works by finding invariant processes for the infinite family of finite-state systems. Intuitively, an invariant process captures the relevant behavior of the entire family. We have used invariant processes to verify a family of token rings and parity trees. We are currently designing a system called Induct that will allow us to verify large examples. Induct will allow the user to describe an infinite family of finite-state systems and a property. Induct will automatically try to produce an invariant process for the described family of finite-state systems. We have also extended our ideas to handle fairness constraints. For example, we may want to verify that a family of token rings treats its com-

ponents “fairly”, i.e., a component requesting a token infinitely often should get the token infinitely often.

5.5.3 Word-level model checking

We have investigated two approaches for formal verification of arithmetic circuits: one based on model checking and the other based on theorem proving. We have developed a concise representation for functions mapping boolean vectors into integers. By combining this representation with symbolic model checking, we have introduced word-level model checking, which technique can be applied to many arithmetic circuits. We have also used our theorem prover, Analytica, for hardware verification. As an example to illustrate the power of these verification techniques, we have verified a division circuit that implements the IEEE floating point standard.

5.6 Interfacing with Java

In [Hardwick and Sipelstein 96] we describe our experiences using Java as an intermediate language for the high-level programming language NESL. First, we describe the design and implementation of a system for translating VCODE — the current intermediate language used by NESL — into Java. Second, we evaluate this translation by comparing the performance of the original VCODE implementation with several variants of the Java implementation. The translator was easy to build, and the generated Java code achieves reasonable performance when using a just-in-time compiler. We conclude that Java is attractive both as a compilation target for rapid prototyping of new programming languages and as a means of improving the portability of existing programming languages.

5.7 Mapping of NESL onto distributed memory multiprocessors

In [Hardwick 96] we describe our work on a new method of implementing irregular divide-and-conquer algorithms in a nested data-parallel language model on distributed-memory multiprocessors. The main features discussed are the recursive subdivision of asynchronous processor groups to match the change from data-parallel to control-parallel behavior over the lifetime of an algorithm, switching from parallel code to serial code when the group size is one (with the opportunity to use a more efficient serial algorithm), and a simple manager-based, runtime load-balancing system. Sample algorithms translated from NESL into C and MPI using this method run significantly faster than the current NESL system and show the potential for further speedup.

5.8 High performance message routing

This portion of the project investigates both theoretical and practical issues of high speed communication among cooperating processors in a parallel computing environment. This challenge has proved one of the greatest in exploiting the power of parallel computing.

5.8.1 All-to-All routing

Several recent papers have proposed or analyzed optimal algorithms to route all-to-all personalized communication (AAPC) over communication networks such as meshes, hypercubes and omega switches. However, the constant factors of these algorithms are often an obscure function

of system parameters such as link speed, processor clock rate, and memory-access time. In [Stricker and Hardwick 96] we investigate these architectural factors, showing the impact of communication style, network routing table, and most importantly, local memory system, on AAPC performance and permutation routing on the Cray T3D.

The fast hardware barriers on the T3D permit a straightforward AAPC implementation using routing phases separated by barriers, which improve performance by controlling congestion. However, we found that a practical implementation was difficult, and the resulting AAPC performance was less than expected. After detailed analysis, several corrections were made to the AAPC algorithm and to the machine's routing table, raising the performance from 41% to 74% of the nominal bisection bandwidth of the network.

Most AAPC performance measurements are for permuting large, contiguous blocks of data (i.e., every processor has an array of P contiguous elements to be sent to every other processor). In practice, sorting and true h - h permutation routing (where $h=n/p \gg 1$, the number of elements per processor) require data elements to be gathered from their source location into a buffer, transferred over the network, and scattered into their final location in a destination array. We obtain an optimal T3D implementation by chaining local and remote memory operations together. We quantify the implementation's efficiency both experimentally and theoretically, using the recently-introduced copy transfer model, and present results for a counting sort based on this AAPC implementation.

5.8.2 Wormhole routing

In [Cole et al. 96] we describe work proving that increasing the queuing capacity of each switch in a wormhole router — so that each physical channel can support up to Q virtual channels — can speed the router by a factor larger than linear in Q . In particular, this work shows that, in a wide variety of networks, a factor of $D^{(1/Q)}$ appears in the time required to route a set of messages, where D is the diameter of the network. This proof confirms earlier work by Dally in which the benefit of virtual channels was demonstrated through simulations.

5.9 Bibliography

- [Beatty and Bryant 94]
Beatty, D.L. and R.E. Bryant.
Formally verifying a microprocessor using a simulation methodology.
In *Proceedings of the 31st Design Automation Conference*. June, 1994.
- [Blelloch 96]
Blelloch, G.E.
Programming Parallel Algorithms.
Communications of the ACM, March, 1996.
- [Blelloch and Greiner 94]
Blelloch, G.E. and J. Greiner.
A parallel complexity model for functional languages.
Technical Report CMU-CS-94-196, Computer Science Department, Carnegie Mellon University,
October, 1994.
- [Blelloch and Greiner 95]
Blelloch, G.E., and J. Greiner.
Parallelism in Sequential Functional Languages.
In *Proceedings of the Symposium on Functional Programming and Computer Architecture*.
June, 1995.
- [Blelloch and Greiner 96]
Blelloch, G.E., and J. Greiner.
A Provable Time and Space Efficient Implementation of NESL.
In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*.
May, 1996.
- [Blelloch and Maggs 96]
Blelloch, G.E. and B. Maggs.
Parallel algorithms.
ACM Computing Surveys, To appear, 1996.
- [Blelloch and Narlikar 94]
Blelloch, G.E. and G. Narlikar.
A comparison of two N-body algorithms.
In *DIMACS Implementation Challenge Workshop*. DIMACS, October, 1994.
- [Blelloch et al. 94a]
Blelloch, G.E., S. Chatterjee, J.C. Hardwick, J. Sipelstein, and M. Zagha.
Implementation of a portable nested data-parallel language.
Journal of Parallel and Distributed Computing, 1994.
- [Blelloch et al. 94b]
Blelloch, G.E., S. Chatterjee, and M. Zagha.
Solving linear recurrences with loop raking.
Journal of Parallel and Distributed Computing, 1994.

[Blelloch et al. 94c]

Blelloch, G.E., C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith, and M. Zagha.
A comparison of sorting algorithms for the connection Machine CM-2.
In *Communications of the ACM*. ACM, 1994.

[Blelloch et al. 94d]

Blelloch, G.E., S. Chatterjee, J.C. Hardwick, M. Reid-Miller, J. Sipelstein, and M. Zagha.
CVL: A C vector library.
Technical Report CMU-CS-93-114, Computer Science Department, Carnegie Mellon University,
February, 1994.

[Blelloch et al. 94e]

Blelloch, G.E., B.M. Maggs, and G.L. Miller.
The hidden cost of low bandwidth communication.
In U. Vishkin (editor), *Developing a Computer Science Agenda for High-Performance Computing*. ACM Press, 1994.

[Blelloch et al. 95a]

Blelloch, G.E., P. Gibbons, and Y. Matias.
Provably Efficient Scheduling for Languages with Fine-Grained Parallelism.
In *Proceedings of the 7th Annual Symposium on Parallel Algorithms and Architectures*.
SPAA, July, 1995.

[Blelloch et al. 95b]

Blelloch, G.E., P. Gibbons, Y. Matias, and M. Zagha.
Accounting for Memory Bank Contention and Delay in High-Bandwidth Multiprocessors.
In *Proceedings of the 7th Annual Symposium on Parallel Algorithms and Architectures*.
SPAA, July, 1995.

[Blelloch et al. 96]

Blelloch, G.E., G.L. Miller, and D. Talmor.
Developing a Practical Projection-Based Parallel Delaunay Algorithm.
In *Proceedings ACM Symposium on Computational Geometry*. May, 1996.

[Bryant 93]

Bryant, R.E.
Symbolic analysis of masks, circuits, and systems.
In *International Conference on Computer Design*. October, 1993.

[Bryant 95a]

Bryant, R.E.
Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification.
In *International Conference on Computer-Aided Design*. November, 1995.

[Bryant 95b]

Bryant, R.E.
Multipliers and Divider: Insights on Arithmetic Circuit Verification.
1995.
To appear in *Computer-Aided Verification*, 1995.

[Bryant 96]

Bryant, R.E.

Bit-Level Analysis of an SRT Divider Circuit.

In *Proceedings of the 33rd Automated Design Conference*. June, 1996.

Previously appeared as Technical Report CMU-CS-95-140, April 1995.

[Bryant and Chen 95]

Bryant, R.E. and Y.-A. Chen.

Verification of arithmetic functions with binary moment diagrams.

In *Proceedings of the 32nd Design Automation Conference*. June, 1995.

Best paper award in the category "Verification, Simulation, and Test". Also appeared as Technical Report CMU-CS-94-160 in May, 1994.

[Bryant and Seger 94]

Bryant, R.E. and C.-J.H. Seger.

Digital circuit verification using partially-ordered state models.

In *International Symposium on Multi-Valued Logic*. May, 1994.

[Bryant et al. 94]

Bryant, R.E., J.D. Tygar, and L.P. Huang.

Geometric characterization of series-parallel variable resistor networks.

IEEE Transactions on Circuits and Systems, November, 1994.

[Burch et al. 94]

Burch, J., E.M. Clarke, D. Long, K. McMillan, and D. Dill.

Symbolic model checking for sequential circuit verification.

In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. IEEE, April, 1994.

[Campos and Clarke 95]

Campos, S., and E.M. Clarke.

Real-Time Symbolic Model Checking for Discrete Time Models.

AMAST Series in Computing: Theories and Experiences for Real-Time System Development.

In T. Rus and C. Rattray,

World Scientific Publishing Company, 1995.

[Campos et al. 94]

Campos, S., E. Clarke, W. Marrero, M. Minea, and H. Hiraishi.

Computing quantitative characteristics of finite-state real-time systems.

In *Proceedings of the 15th IEEE Real-Time Systems Symposium*. IEEE, December, 1994.

Also available as Technical Report CMU-CS-94-147.

[Campos et al. 95a]

Campos, S., E. Clarke, W. Marrero, and M. Minea.

Timing Analysis of Industrial Real-Time Systems.

In *Workshop on Industrial Strength Formal Specification Techniques*. 1995.

[Campos et al. 95b]

Campos, S., E.M. Clarke, W. Marrero, M. Minea, and H. Hiraishi.

Temporal Verification of Real-Time Systems.

In *IEICE Transactions on Information and Systems*. IEICE, July, 1995.

[Campos et al. 95c]

Campos, S., E.M. Clarke, W. Marrero, and M. Minea.
Versus: a Tool for Quantitative Analysis of Finite-State Real-Time Systems.
In *Workshop on Languages, Compilers, and Tools for Real-Time Systems*. 1995.

[Chen and Bryant 96]

Chen, Y.-A., and R.E. Bryant.
ACV: An Arithmetic Circuit Verifier.
In *Proceedings of International Conference on Computer-Aided Design*. ICCAD, November, 1996.

[Clarke and Zhao 93]

Clarke, E.M. and X. Zhao.
Analytica: a theorem prover for mathematica.
*The Mathematica Journal*3(1):56-71, 1993.

[Clarke and Zhao 94]

Clarke, E. and X. Zhao.
Combining symbolic computation and theorem proving: some problems of Ramanujan.
Automated Deduction—CADE-12, Lecture Notes in Computer Science 814.
In A. Bundy,
Springer-Verlag, 1994.

[Clarke et al. 93]

Clarke, E.M., I.A. Draghicescu, and R.P. Kurshan.
A unified approach for showing language containment and equivalence between various types of ω -automata.
*Information Processing Letters*46:301-308, 1993.

[Clarke et al. 94a]

Clarke, E., O. Grumberg, and K. Hamaguchi.
Another look at LTL model checking.
In *Proceedings of Conference on Computer-Aided Verification*. CAV, June, 1994.

[Clarke et al. 94b]

Clarke, E., O. Grumberg, and D. Long.
Verification tools for finite-state concurrent systems.
A Decade of Concurrency - Reflections and Perspectives, Lecture Notes in Computer Science 803.
In J.W. deBakker, W.P. deRoeve, and G. Rozenberg,
Springer-Verlag, 1994.

[Clarke et al. 94c]

Clarke, E.M., O. Grumberg, K. McMillan, and X. Zhao.
Efficient generation of counterexamples and witnesses in symbolic model checking.
Technical Report CMU-CS-94-204, Computer Science Department, Carnegie Mellon University,
October, 1994.
Also accepted by DAC95, June 1995.

- [Clarke et al. 95a]
Clarke, E.M., O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K. McMillan, and L.A. Ness.
Verification of the Futurebus+ Cache Coherence Protocol.
Formal Methods in Systems Design 6:217-232, 1995.
- [Clarke et al. 95b]
Clarke, E.M., T. Filkorn, and S. Jha.
Exploiting Symmetry in Temporal Logic Model Checking.
Formal Methods in System Design, 1995.
To appear.
- [Clarke et al. 95c]
Clarke, E.M., O. Grumberg, and D.E. Long.
Model Checking.
Lecture Notes in Computer Science.
Springer Verlag, 1995.
- [Cole et al. 96]
Cole, R.J., B.M. Maggs, and R.K. Sitaraman.
On the benefit of supporting virtual channels in wormhole routers.
In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*.
ACM, June, 1996.
- [Feldmann et al. 95]
Feldmann, A., B.M. Maggs, J. Sgall, D.D. Sleator, and A. Tomkins.
Competitive Analysis of Call Admission Algorithms That Allow Delay.
Technical Report CMU-CS-95-102, Computer Science Department, Carnegie Mellon University,
January, 1995.
- [Fisher and Ghuloum 94]
Fisher, A.L. and A. Ghuloum.
Parallelizing complex scans and reductions.
In *Proceedings of the 1994 ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 1994.
- [Ghosh et al. 95]
Ghosh, B., F.T. Leighton, B.M. Maggs, S. Muthukrishnan, C.G. Plaxton, R. Rajaraman, A.W. Richa, R.E. Tarjan, and D. Zuckerman.
Tight Analyses of Two Local Load Balancing Algorithms.
In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*. ACM, May, 1995.
- [Greiner 94]
Greiner, J.
A comparison of data-parallel algorithms for connected components.
In *Proceedings of ACM SIGPLAN Symposium on Parallel Algorithms and Architectures*.
ACM, June, 1994.

- [Greiner and Bluelloch 95]
Greiner, J., and G.E. Bluelloch.
Connected Components Algorithms.
In G.W. Sabot (editor), *High Performance Computing: Problem Solving with Parallel and Vector Architectures*. Addison Wesley, Reading, MA, 1995.
- [Greiner and Bluelloch 96]
Greiner, J., and G.E. Bluelloch.
A Provably Time-Efficient Parallel Implementation of Full Speculation.
In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 309-321. January, 1996.
- [Gremban et al. 95]
Gremban, K.D., G.L. Miller, and M. Zagha.
Performance Evaluation of a New Parallel Preconditioner.
In *Proceedings of the 9th International Parallel Processing Symposium*. IPPS, April. 1995.
- [Gupta and Fisher 93]
Gupta, A. and A.L. Fisher.
Representation and symbolic manipulation of linearly inductive boolean functions.
In *International Conference on Computer-Aided Design*. November, 1993.
- [Hardwick 94]
Hardwick, J.C.
Porting a vector library: a comparison of MPI, Paris, CMMD and PVM.
In *Proceedings of Scalable Parallel Libraries Conference*. SPLC, October, 1994.
- [Hardwick 96]
Hardwick, J.C.
An Efficient Implementation of Nested Data Parallelism for Irregular Divide-and-Conquer Algorithms.
In *First International Workshop on High-Level Programming Models and Supportive Environments*. April, 1996.
- [Hardwick and Sipelstein 96]
Hardwick, J.C., and J. Sipelstein.
Java as an Intermediate Language.
Technical Report CMU-CS-96-161, Computer Science Department, Carnegie Mellon University,
August, 1996.
- [Jain and Bryant 93]
Jain, A. and R.E. Bryant.
Inverter minimization in logic networks.
In *International Conference on Computer-Aided Design*. November, 1993.
- [Jain et al. 96]
Jain, A., K.A. Nelson, and R.E. Bryant.
Verifying Nondeterministic Implementations of Deterministic Systems.
In *Formal Methods in Computer-Aided Design*. FMCAD, November, 1996.

[Leighton and Maggs 95]

Leighton, F.T. and B.M. Maggs.

Fast algorithms for finding $O(\text{congestion} + \text{dilation})$ packet routing schedules.

In *Proceedings of the 28th Hawaii International Conference on System Sciences*. HICSS, January, 1995.

[Long et al. 94]

Long, D.E., A. Browne, E.M. Clarke, S. Jha, and W.R. Marrero.

An improved algorithm for the evaluation of fixpoint expressions.

In *Proceedings of Conference on Computer-Aided Verification*. CAV, June, 1994.

[Maggs et al. 94a]

Maggs, B.M., F.T. Leighton, and S.B. Rao.

Packet routing and job-shop scheduling in $O(\text{congestion} + \text{dilation})$ steps.

Combinatorica 14(2):167-180, 1994.

[Maggs et al. 94b]

Maggs, B.M., F.T. Leighton, S.B. Rao, and A.G. Ranade.

Randomized routing and sorting on fixed-connection networks.

Journal of Algorithms 17(1):157-205, 1994.

[Maggs et al. 95]

Maggs, B.M., L.R. Matheson, and R.E. Tarjan.

Models of parallel computation: a survey and synthesis.

In *Proceedings of the 28th Hawaii International Conference on System Sciences*. HICSS, January, 1995.

[Pandey et al 96]

Pandey, M., R. Raimi, D.L. Beatty, and R.E. Bryant.

Formal Verification of PowerPC(TM) Arrays using Symbolic Trajectory.

In *Proceedings of the 33rd Automated Design Conference*. June, 1996.

[Reid-Miller 94]

Reid-Miller, M.

List-ranking and list-scan on the Cray C90.

In *Proceedings of ACM SIGPLAN Symposium on Parallel Algorithms and Architectures*.

ACM, June, 1994.

Also available as Technical Report CMU-CS-94-101, and to appear in *Journal of Computer and System Sciences*.

[Seger and Bryant 95]

Seger, C.-J.H., and R.E. Bryant.

Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories.

Formal Methods in System Design 6(2):147-190, 1995.

[Sheffler and Bryant 93]

Sheffler, T.J. and R.E. Bryant.

An analysis of hashing on parallel and vector computers.

In *International Conference on Parallel Processing*. August, 1993.

[Starkey and Bryant 95]

Starkey, M., and R.E. Bryant.

Using Ordered Binary-Decision Diagrams for Compressing Images and Image Sequences.

Technical Report CMU-CS-95-105, Computer Science Department, Carnegie Mellon University,

January, 1995.

[Stricker and Hardwick 96]

Stricker, T.M., and J.C. Hardwick.

From AAPC Algorithms to High Performance Permutation Routing and Sorting.

In Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures. June, 1996.

6. Intelligent Information Integration

The objective of this research has been to develop a network of software agents that provide information access and fusion from heterogeneous information sources and to experiment with these agents within a fielded system for supporting a collection of office work tasks. Our thesis has been that methods for learning and negotiation will dramatically improve the effectiveness, robustness, scalability, and maintainability of such systems.

Our accomplishments include development of a knowledge-based assistant for calendar management that learns scheduling preferences of its users, and integration of this with two other agents (a personnel agent and visitor hosting agent) to assist jointly in hosting visitors. We subsequently began a new thrust to explore the role of machine learning for information access on the worldwide network. More recently, a significant portion of our effort has been devoted to development and initial deployment of a newsgroup reader that learns user reading interests so that it can retrieve new articles of high relevance, and a tour guide agent for the world wide web that learns to suggest which hyperlinks to follow.

6.1 System design

Our strategy has been to develop a collection of learning and negotiating mediators that are hierarchically organized. Characteristics of the architecture are:

- *Sharability* — mediators are sharable by many user applications
- *Flexibility* — mediators can interact in new configurations "on-demand", depending on the information requirements of a particular decision making task
- *Modularity* - mediators are kept simple for ease of maintenance, and their results are composable.

Mediators are of two types:

- *Information Assistant* mediators, which oversee specific information sources and interface them to other mediators
- *Task Assistant* mediators, each of which has a model of a specific task. Task Assistants interface users to the collection of other mediators.

Mediators in the system learn in several ways.

- They learn regularities implicit in the databases. These learned regularities are used to:
 - Infer missing data values
 - Identify irregular data entries
 - Explicitly communicate general, learned relationships to other software agents and to users.
- They act as "learning apprentices" that observe user actions and unobtrusively and incrementally learn models of the user's preferences, user's task and users' work processes, thus being able to adapt to the user's mode of interaction, anticipate a user's information retrieval needs and respond quickly to unexpected events.
- They learn effective information retrieval and negotiation strategies. In particular, they learn how to resolve information conflicts, which information mediators are capable of answering which specific queries, with what reliability, what cost, and under what circumstances.

The mediators in the system are distributed and negotiate in order to:

- Resolve conflicts in the retrieved information
- Achieve agreement on domain decisions, taking into consideration the preferences of their users.

Negotiation is supported by learned knowledge (e.g. regularities in different data bases, user's task and user's interaction model) and in return serve as sources of additional experience for further learning.

6.2 Significance of task assistants

Our research has significantly improved the ability of computer users to access and fuse information from heterogeneous sources, and has lowered the barrier to developing software systems customized to specific users and application tasks. In particular:

- Self-customizing software agents based on machine learning methods will reduce by an order of magnitude the user-specific effort required to convert generic application software (e.g., schedulers) into useful systems for specific tasks and users.
- User transparent information access and integration in user-specific terms, irrespective of information source location and format, will significantly increase the capability of planners. This is especially important in crisis action planning.
- Automated support for interleaving of planning and information retrieval for users distributed in space and time will reduce by an order of magnitude cycle time for concurrent planning or design.

Our approaches to learning and negotiation offer the possibility to significantly improve the robustness and effectiveness of Intelligent Information Integration systems. For example a learning information filter or tour guide could be of significant use for monitoring the flood of heterogeneous streams of information for intelligence surveillance.

6.3 Developing task assistants: the Pleiades system

We have developed a collection of mediator Task Assistants, collectively called the Pleiades system. Prototype or mature versions have been demonstrated of CAP (our calendar apprentice, which learns user scheduling preferences), an electronic news reader that learns user interests in reading news from the net, a Visitorhost that helps schedule visitors for technical briefings, and a Personnel Information mediator that provides information about specific individuals and their jobs. Machine learning capabilities have been demonstrated in the Calendar Apprentice. Collaboration between the Visitorhost and Personnel mediator has been demonstrated in the context of identifying relevant information for scheduling technical briefings at Carnegie Mellon.

We enhanced the Pleiades collection of interoperating agents with additional Information Assistants and Task Assistants to support an extended visitor hosting scenario using additional Internet-based information sources and services. The current Information and Task Assistants that are used in the extended visitor hosting scenario are:

Information-Specific Agents

1. *Finger* agent, which heuristically parses the retrieved information from remotely residing "finger" data bases. The possible types of information that can be acquired in this way include: work title, research interests, work and home phone numbers, vacation plan, etc.
2. *Who's-Who* agent, which accesses on-line Carnegie Mellon who's who database through http-based queries. The fields in the database include: name, title, affiliation, campus office, campus phone number, home address and e-mail address.
3. *Faculty Interests* agent, which can be used to retrieve information about the faculty members in the School of Computer Science at Carnegie Mellon with respect to their research interests.
4. *Computer-Science-Directory* agent, which can get the information about phone number, office number, home address, etc. for all the members of the School of Computer Science at Carnegie Mellon, including faculty members, staff and students.
5. *Sean* communication agent that routes messages to appropriate agents and translates the messages into the recipient's format to ensure agent interoperability.

Task-Specific Agents

1. *Host-Visitor* agent, which accepts input from the user regarding the visitor's identity, research interests, tentative meeting date and meeting durations. The Host-Visitor interacts with other agents to arrange the visitor's meetings and schedule at Carnegie Mellon.
2. *CAP* personalized calendar management apprentice, which provides an editing interface to a calendar, and learns users' scheduling preferences by generalizing from specific observed events that appear on the calendar.
3. *Scheduling* agent, which takes the responsibility of maintaining a visitor's meeting schedule. The Scheduling agent coordinates with personalized meeting calendar apprentices, such as CAP that users may possess, or exchanges e-mail with users directly, to find out whether they agree to meet with the visitor and to resolve any scheduling conflicts.
4. *Personnel Finder* agent, which coordinates all personnel information agents (e.g., Finger, Who's Who, etc.), and resolves any conflicting information returned by them. Personnel Finder also accesses resources on the Internet to find information about people.
5. *Interface* agent, which presents to the user acquired information from task or information specific agents.
6. *Information Flow Visualization* agent that monitors the information flow among the rest of the agents in the visitor hosting scenario and dynamically visually displays the flow on a separate screen. This agent has been very useful in visualizing, understanding and debugging the distributed agent interactions in Pleiades.

Our Personnel Information mediator provides mediated interoperation between the Visitorhost task mediator and a number of data bases. It (1) allows posting of queries in domain-specific terms, (2) frees the requester from having to know where the information is stored or how it is organized, and (3) returns useful information, even when the databases are incomplete.

Our Visitorhost task mediator: (1) has a task model defining the terminology for posing queries,

(2) interacts with the Personnel Information mediator to get information from a variety of information sources, and (3) reformulates answers to queries as new queries, thus interleaving inference and information retrieval.

We completed experiments with the learning Calendar Apprentice, based on five user-years of fielded use by a handful of routine users. These experiments indicated that the system could learn automatically thousands of user-specific rules that captured the scheduling preferences of various individuals. Results of these experiments are reported in [Mitchell et al. 94].

6.4 Learning reading interests from experience

We produced an electronic newsreader (NewsWeeder) that learns users reading interests from experience. It uses these interests to create an additional personalized newsgroup containing the most relevant new articles from a broad range of newsgroups that the user otherwise does not see. Results show that the learned user profile successfully increases the proportion of articles the user finds interesting. These results, summarized in [Lang 95], were presented in July 1995 to the Machine Learning conference. A major development in this project was the development of a new algorithm for learning to classify text. In contrast to the most common algorithm in the information retrieval field, TFIDF, this new algorithm is based on a minimum description length (MDL) approach. In this approach, the program first learns the probabilities of occurrence of individual words, conditioned on the class from which the article is drawn. These probabilities allow estimating the minimum coding length needed when compressing a new article from the same class. A new article is then classified by assigning it to the class that best compresses it. In NewsWeeder experiments this MDL approach significantly outperformed TFIDF.

6.5 An experience-assisted web agent

We developed an agent to assist users in locating information on the world wide web. The agent (WebWatcher) assists users by accompanying them from page to page, highlighting useful links and suggesting new pages to visit. Its advice is learned automatically from experience, by observing the response of previous users with similar interests. Over the past six months we have further developed the system software and explored alternative approaches to learning advice-giving strategies from experiments. At the end of this reporting period, WebWatcher was developed fully enough to enable its first deployment on a large scale. We deployed it on the front door Web page of Carnegie Mellon's School of Computer Science, where it served approximately 2000 users during its first few weeks, and many thousands of users since its inception.

We have received numerous inquiries asking for technical details of this system. See <http://www.cs.cmu.edu/~webwatcher> for more information.

6.6 Warren: A portfolio management system

We developed a prototype system, Warren, comprised of a collection of distributed software agents that aid users in portfolio investment management by accessing, filtering, and integrating information from the World Wide Web. The overall portfolio management task has several component tasks. These include eliciting (or learning) user profile information, collecting infor-

mation on the user's initial portfolio position, and suggesting and monitoring a re-allocation to meet the user's current profile and goals. As time passes, assets in the portfolio will no longer meet the user's needs (and these needs may also be changing as well). Our initial system focuses on this ongoing portfolio monitoring process. See <http://www.cs.cmu.edu/~softagents>, a general page on our software agents, including the Warren.

6.7 Bazaar: A formal negotiation model

We undertook development of a formal negotiation model, called Bazaar, based on bargaining theory for exploration of multiagent, multiissue, incremental negotiation strategies with limited information. This model also constitutes a framework within which we are studying multiagent learning issues. We have already obtained some initial theoretical results (proving that under certain assumptions, learning is beneficial in a multiagent setting). We have also performed experiments in a simulated negotiation setting. In our experiments, a *nonlearning* agent, makes decisions based solely on its own reservation price. A *learning* agent makes decisions based on both its own and its opponent's reservation price. Note that reservation prices are private information and there is no way that an agent can know the exact value of its opponent's reservation price, even after an agreement has been reached. However, each learning agent can have some a priori estimation of its opponent's reservation price and update its estimation during the negotiation process using a Bayesian belief updating mechanism. We measured the quality of a particular bargaining process using the normalized joint utility fashioned after the Nash solution. Our results show that learning is beneficial for the learning agent(s) both in terms of contract quality and computational efficiency.

6.8 A World Wide Knowledge Base

In the latter part of this reporting period, we initiated a World Wide Knowledge Base project. This is a new effort to develop a software agent to construct a symbolic, probabilistic, knowledge base automatically by exploring and mining the World Wide Web. The specific approach is to extend our earlier machine learning approaches to provide an agent that can be trained to extract knowledge to fit a user-provided ontology that defines the classes of objects and relations of interest.

Preliminary experiments were promising. Given an ontology that defines knowledge base classes including PERSON, STUDENT, FACULTY, RESEARCH-PROJECT, UNIVERSITY-COURSE, and DEPARTMENT, we trained the agent to recognize Web pages for each of these classes. After training on several thousand pages taken describing three universities, we gave it pages from a fourth university. Of the 555 pages describing instances of STUDENT, it correctly extracted 504 of these and added them to its knowledge base as new students. Similar results were obtained for other classes in the ontology. Further details are available in [Craven et al 97].

6.9 Other initiatives

6.9.1 Active data management

We have developed information agents that perform sophisticated *active data management* functions for numeric and textual data. An agent advertises its active data management capabilities to a matchmaker agent, who can be queried about the availability of such services by requesting agents (its customers). A customer registers with the agent(s) providing the services the customer needs, receiving notification whenever the corresponding information source is updated. Registration for information services may include reporting deadlines. For example, a monitoring request to the Security APL ticker tracker agent may be "notify me, within 5 minutes of the occurrence of the event, when the price of IBM increases by 10%."

6.9.2 Matchmaker agents

We have developed *matchmaker* agents, information agents that know about capabilities and services that can be provided by other agents. The matchmaker is a type of yellow pages that can be queried for particular services by other agents. When an agent is started, it first registers, or advertises its capabilities, with the matchmaker.

6.9.3 Information services

We have developed a language to describe information services, and protocols for advertisement of information services and registration with agents providing such services.

6.9.4 Self-monitoring

When an information agent detects that its performance in providing information services is degrading (e.g., it detects missing reporting deadlines), it can now automatically clone itself and divide its customer list among its clones to increase quality of service.

6.9.5 Reusable agent architecture

Our work has resulted in a generic and reusable agent architecture that facilitates rapid agent construction. This architecture consists of:

- *Planning component:* The planning module takes as input a set of goals, and then produces a plan that satisfies the goals. The planning module of the task agents can be a full-fledged planner, whereas the planning module of the interface agents and the information agents is much simpler—consisting of retrieval and instantiation of plan templates.
- *Scheduling module:* This module schedules each of the plan steps. The agent scheduling process in general takes as input the agent's current set of plan instances (in particular, the set of all executable actions) and decides which action, if any, is to be executed next. This action is then identified as a fixed intention until it is actually carried out (by the execution component). While task agents can require very sophisticated scheduling, in our initial implementation of information agents we use a simple, earliest-deadline-first schedule execution heuristic.

- *Execution Monitoring:* The agent execution monitoring process takes as input the agent's next intended action and prepares, monitors, and completes its execution. The execution monitor prepares an action for execution by setting up a context (including the results of previous actions, etc.) for the action. It monitors the action by optionally providing the associated computation limited resources—for example, the action may be allowed only a certain amount of time and if the action does not complete before that time is up, the computation is interrupted and the action is marked as having failed.
- *Communication and Coordination:* This component prepares messages to be sent out, interprets received messages, and reasons about to whom to send a request, and when. The agents communicate through the KQML language.
- *Plan library:* This library contains skeletal plans and plan fragments that are indexed by goals, and can be retrieved and instantiated according to the current input parameters. The retrieved and instantiated plan fragments are used to form the agent's task tree that is incrementally executed.
- *Belief and facts data base:* These structures contain facts and other knowledge related to the agent's functionality.

6.9.6 Interest and opportunity matching

We have developed an agent that learns user research interests and notifies users of conference announcements that are available on newsgroups, and requests for proposals available from the Electronic Commerce Business Daily. The agent's learning is "bootstrapped" using research papers and reports the user has written. Two technologies were examined and experimentally compared: information filtering and neural networks.

6.10 Bibliography

[Armstrong et al. 95]

Armstrong, R., D. Freitag, T. Joachims, and T. Mitchell.
WebWatcher: A Learning Apprentice for the World Wide Web.
In *Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*. AAAI, March, 1995.

[Bocionek and Mitchell 93]

Bocionek, S. and T. Mitchell.
Office automation systems that are programmed by their users.
In *Proceedings of the 23rd Annual Conference of the German Association of Computer Science (Gesellschaft fur Informatik)*. GI, September, 1993.

[Caruana and Freitag 94]

Caruana, R. and D. Freitag.
Greedy attribute selection.
In *The Proceedings of the Eleventh International Conference on Machine Learning*. CML, 1994.

[Craven et al 97]

Craven, M., D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and C.Y. Quek.
Learning to Extract Symbolic Knowledge from the World Wide Web.
In *International Conference on Machine Learning*. ICML'97, 1997.
Submitted.

[de Kroon et al. 96]

de Kroon, H.C.M., T.M. Mitchell, and E.J.H. Kerckhoffs.
Improving Learning Accuracy in Information Filtering.
In *Workshop on HCI and ML, in affiliation with the 1996 International Conference on Machine Learning*. ICML, July, 1996.

[Decker et al 97]

Decker, K., A. Pannu, K. Sycara, and M. Williamson.
Designing Behaviors for Information Agents.
In *Proceedings of the First International Conference on Autonomous Agents*. ICAA, February, 1997.

[Decker et al. 96a]

Decker, K., K. Sycara, and M. Williamson.
Intelligent Adaptive Information Agents.
In *Proceedings of the AAAI-96 Workshop on Intelligent Adaptive Agents*. AAAI, August, 1996.

[Decker et al. 96b]

Decker, K., K. Sycara, and D. Zeng.
Designing a Multi-Agent Portfolio Management System.
In *Proceedings of the AAAI-96 Workshop on Internet-Based Information Systems*. AAAI, August, 1996.

[Joachims et al 97]

Joachims, T., D. Freitag, and T. Mitchell.
WebWatcher: A Tour Guide for the World Wide Web.
In Proceedings of the 1997 IJCAI. IJCAI, 1997.

[Joachims et al. 95]

Joachims, T., T. Mitchell, R. Armstrong, and D. Freitag.
WebWatcher: Machine Learning and Hypertext.
In German Workshop on Machine Learning. August, 1995.

[Lang 95]

Lang, K.
A newsreader that learns the interests of its users.
In Proceedings of the International Conference on Machine Learning 1995. ICML, July, 1995.

[Lewis and Sycara 93]

Lewis, M. and K. Sycara.
Informed decision making in multi-specialist cooperation.
*Group Decision and Negotiation*2(3), 1993.

[Liu and Sycara 94]

Liu, J.S. and K. Sycara.
Distributed constraint-directed meeting scheduling.
In Workshop Notes of the CAIA-94 Workshop on Coordinated Design and Planning. CAIA, March, 1994.

[Liu and Sycara 95]

Liu, J.S. and K. Sycara.
Exploiting Problem Structure for Distributed Constraint Optimization.
In First International Conference on Multi Agent Systems. ICMAS, June, 1995.

[Mitchell et al. 94]

Mitchell, T., R. Caruana, D. Freitag, J. McDermott, and D. Zabowski.
Experience with a learning personal assistant.
*Communications of the ACM*37(7):81-91, 1994.

[Miyashita and Sycara 94]

Miyashita, K. and K. Sycara.
A framework for case-based revision for schedule generation and reactive schedule management.
*The Journal of the Japanese Society of Artificial Intelligence*9(3):426-435, 1994.

[Miyashita and Sycara 95a]

Miyashita, K. and K. Sycara.
CABINS: a framework of knowledge acquisition and iterative revision for schedule optimization and reactive repair.
AI Journal, 1995.
To appear.

- [Miyashita and Sycara 95b]
Miyashita, K. and K. Sycara.
Improving System Performance in Case-Based Iterative Optimization through Knowledge Filtering.
In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.
IJCAI-95, August, 1995.
- [Pannu and Sycara 96]
Pannu, A., and K. Sycara.
A Personal Text Filtering Agent.
In *Working Notes of the AAAI Spring Symposium Machine Learning for Information Access*.
AAAI, 1996.
- [Sycara 93]
Sycara, K.
Machine learning for intelligent support of conflict resolution.
Decision Support Systems 10:121-136, 1993.
- [Sycara and Liu 94]
Sycara, K. and J.S. Liu.
Distributed meeting scheduling.
In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. CSS,
August, 1994.
- [Sycara and Miyashita 94]
Sycara, K. and K. Miyashita.
Case-based acquisition of user preferences for solution improvement in ill-structured domains.
In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI, July,
1994.
- [Sycara and Miyashita 95]
Sycara, K. and K. Miyashita.
Learning Control Knowledge through Case-Based Acquisition of User Optimization Preferences.
In Y. Kodratoff and G. Tecuci (editor), *Knowledge Acquisition and Machine Learning: An Integrated Approach*. Morgan Kaufmann Publishers, 1995.
- [Sycara and Zeng 94]
Sycara, K. and D. Zeng.
Towards an intelligent electronic secretary.
In *Proceedings of the International Conference on Information and Knowledge Management Workshop on Intelligent Information Agents*. CIKM-94, December, 1994.
- [Sycara and Zeng 95]
Sycara, K. and D. Zeng.
Task-based multiagent coordination for information gathering.
In C. Knoblock and A. Levy (editor), *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*. AAAI, 1995.

[Sycara and Zeng 96a]

Sycara, K., and D. Zeng.

Coordination of Multiple Intelligent Software Agents.

International Journal of Cooperative Information Systems, 1996.

To appear.

[Sycara and Zeng 96b]

Sycara, K., and D. Zeng.

Multi-Agent Integration of Information Gathering and Decision Support.

In Proceedings of the 12th European Conference on AI. ECAI, August, 1996.

[Sycara and Zeng 96c]

Sycara, K., and D. Zeng.

Coordination of Multiple Intelligent Software Agents.

International Journal of Intelligent and Cooperative Information Systems, 1996.

[Sycara and Zeng 96d]

Sycara, K. and D. Zeng.

Coordination of Multiple Intelligent Software Agents.

International Journal of Intelligent and Cooperative Information Systems 5(2-3):181-211, 1996.

[Sycara et al 96]

Sycara, K., K. Decker, A. Pannu, M. Williamson, and D. Zeng.

Distributed Intelligent Agents.

IEEE Expert: Intelligent Systems and their Applications 11(6), 1996.

[Sycara et al. 95]

Sycara, K., D. Zeng, and K. Miyashita.

Using case-based reasoning to acquire user scheduling preferences that change over time.

In Proceedings of the Eleventh International Conference on Artificial Intelligence Applications (CAIA '95). IEEE, February, 1995.

[Williamson et al. 96]

Williamson, M., K. Decker, and K. Sycara.

Unified Information and Control Flow.

In Proceedings of the AAAI-96 Workshop on Theories of Action, Planning and Control: Bridging the Gap. AAAI, August, 1996.

[Zeng and Sycara 94]

Zeng, D. and K. Sycara.

Preliminary report on generic negotiator.

In Proceedings of the International Conference on Information and Knowledge Management Workshop on Intelligent Information Agents. CIKM-94, December, 1994.

[Zeng and Sycara 96]

Zeng, D., and K. Sycara.

Bayesian Learning in Negotiation.

In Proceedings of the AAAI Stanford Spring Symposium on Adaptation, Co-evolution and Learning in Multi-Agent Systems. AAAI, March, 1996.

[Zeng and Sycara 97]

Zeng, D. and K. Sycara.

Baysean Learning in Negotiation.

International Journal of Human Machine Systems, 1997.

To appear.

7. Spoken-language systems

The long-term goal of Carnegie Mellon's speech recognition effort continues to be the enhancement of the accuracy, robustness, portability, scalability and utility of spoken language systems, through the development of strategies to automatically acquire knowledge at all levels of the process and through unification of structures to represent this knowledge. The goal of the effort reported here has been to create technology for real-time unlimited vocabulary spoken language processing in the context of practical applications.

7.1 Practical applications of speech recognition research

Carnegie Mellon pursues a broad program of research in the context of speech applications that offer practical value to the Department of Defense. Three systems developed under the current contract draw upon this work. Each application emphasizes a currently difficult aspect of speech technology:

- A speech interface for a wearable computer provides access to information resources for the mobile warfighter.
- The News-on-Demand system uses speech recognition for automatic indexing of broadcast materials and for subsequent retrieval.
- DIPLOMAT provides a flexible, multimodal, speech-to-speech translation system.

The wearable speech-system work focuses on developing speech-only interfaces to small, wearable devices for which conventional interface modalities are inappropriate. Such systems require the development of strategies for interacting fluently with the user, providing for orientation and error correction. We first addressed this problem in the context of an amphibious assault vehicle inspection task for the Marines, where speech is the means of recording inspection data and of accessing maintenance resources. Our current work broadens this focus to the DIPLOMAT system and information access, where speech is used potentially in conjunction with other input modalities in a reconfigurable multimedia interface.

The News-on-Demand work concentrates on the problem of decoding "found speech," that is, speech not originally produced with the intent of being automatically decoded. This domain requires the capability to automatically segment a broadcast stream and to automatically adapt the language model to content. It emphasizes basic recognition techniques and adaptation to evolving situations. The testbed for News-on-Demand is the indexing of broadcast news, with a goal for increasing accuracy of transcription and the sophistication of queries that are possible on the resulting database.

The DIPLOMAT work, reported separately in Section 12, addresses two issues: cross-language communication among a diverse range of individuals and procedures for rapidly deploying speech and translation capabilities. This domain emphasizes speech interfaces that are simple to use and that can be learned easily. It also emphasizes procedures for rapidly acquiring and processing language-specific information (needed for acoustic, lexical, and language modeling of speech and for creating translation knowledge-bases, both transfer-based and example-based). The testbed for DIPLOMAT is rapid development of speech-to-speech translation capabilities for a series of nonEnglish languages.

The application-specific work described above draws on a core of spoken-language research carried out by Carnegie Mellon. The goal of this more fundamental work is to:

- Improve speech recognition through fundamental advances in technology, including automatic acquisition of phonetic, lexical, syntactic, and semantic knowledge.
- Develop dynamic, domain-language adaptation
- Develop algorithms to detect and assimilate new words and to extend grammatical coverage for them
- Develop algorithms for environmental robustness to maintain good recognition performance under varied conditions.

7.2 Accomplishments

Carnegie Mellon's Speech Group has substantially advanced the state of the art in speech recognition, spoken-language understanding, and dialog modeling. Several concept-demonstration systems embody these advances, and our work is available (in software form) to the community.

- We extended the DIPLOMAT cross-lingual speech communications system to work with two additional languages of interest to DoD, Haitian Creole and Korean. Generalized and improved rapid deployment techniques.
- We extended our mobile/wearable speech systems work to a new domain, indexing (e.g., license-plate lookup) system developed in cooperation with the City of Pittsburgh Police. Extensions included a more sophisticated dialog-level model for interaction and integral use of speech response.
- Delivered and installed a version of the News on Demand system in the DARPA TIE facility. Significantly improved retrieval accuracy for News on Demand, by tailoring retrieval to the characteristics of the domain (errorful transcribed speech). Accurate retrieval is possible even with significant degradation in transcription accuracy.
- We developed techniques for adaptive language-modeling based on small amounts of data and were among the first sites to pursue active research in this area.
- We developed techniques for dynamically modifying running decoders, including changes to lexicon and language model.
- We developed techniques for integrating speech-recognition and understanding and demonstrated significant error reduction (20%) based on these techniques.
- Carnegie Mellon conceived and developed a broadcast-news component (News-on-Demand) for the Informedia system with an emphasis on rapid access to broadcast materials.
- Applying speech technology to support retrieval of spoken-language information, we developed techniques for annotating recognizer output with probabilities of correct recognition and using this information to improve retrieval performance.
- Continued development of our wearable system, which incorporates advanced recognition technology, led to new paradigms for speech-only interaction, and we ported the system to a new domain (law enforcement).
- We developed procedures that use speech understanding to aid in completing online, electronic forms and investigated the structure of generalized interfaces to Web material. Our demonstration system embodied techniques that interpret HTML code to aid recognition and enhance Web-based forms to facilitate spoken-language interaction.

- We further refined our techniques for speech-based Web-browsing, an approach that utilizes plug-in, Java, and server models. We also developed several Web-based services to support speech-system development, using them internally and making them openly available on the Web.

7.3 Technology transition

Carnegie Mellon provides a suite of speech-recognition tools for use by technologically sophisticated research and development organizations as a point of departure for research and productization efforts. We have migrated our software from a research environment to successively more accessible development environments, currently including Windows and Visual Basic. The suite currently includes tools for acoustic modeling, language modeling, a decoder, a spoken-language parser and example applications. We also provide (in the public domain) a 100,000+ word pronouncing dictionary used for lexical modeling and implementations of speech coding algorithms.

The Speech Library has been used (and productized) by Apple, DEC, IBM, Kurzweil AI, Microsoft, Verbex, VPC, and Sun. Impact is measured by the availability of products from computer manufacturers (such as IBM and Apple) based on Carnegie Mellon technology. At the same time, we provide our software to universities, federal contractors (in the past year, Lockheed Martin and SAIC) and government labs for research into speech interfaces and for the development of prototype systems. In the past year we have provided systems to NIST, NRL, and the U.S. Army Corps of Engineers.

A significant portion of our effort has been devoted to developing resources that serve the community at large:

- With Cambridge University, we produced a second version of our Spoken Language Modeling Toolkit, the first such freely available resource for language modeling and one that sees wide use throughout the community.
- The Sphinx-II Recognition Toolkit provides a full range of tools for modeling and developing speech-systems. It has been made available to both research and industrial users.

7.4 Bibliography

[Hauptmann et al. 98]

Hauptmann, A.G., R.E. Jones, K. Seymore, M.A. Siegler, S.T. Slattery, and M.J. Witbrock. Experiments in Information Retrieval from Spoken Documents.

In *Proceedings of the Broadcast News Transcription and Understanding Workshop (BNTUW-98)*. DARPA, February, 1998.

Lansdowne, VA.

This paper describes the experiments performed as part of the TREC-97 Spoken Document Retrieval Track. The task was to pick the correct document from 35 hours of recognized speech documents, based on a text query describing exactly one document. Among the experiments we described here are: Vocabulary size experiments to assess the effect of words missing from the speech recognition vocabulary; experiments with speech recognition using a stemmed language model; using confidence annotations that estimate of the correctness of each recognized word; using multiple hypotheses from the recognizer. And finally we also measured the effects of corpus size on the SDR task. Despite fairly high word error rates, information retrieval performance was only slightly degraded for speech recognizer transcribed documents.

8. Sharing Viewpoints, Objects, and Animations

Our objectives for this effort were to:

- Determine cost effective delivery mechanisms of graphics for visual and collaborative techniques
- Develop a predictive model for when a head mounted display (HMD), CAVE, large screen display, or traditional desktop displays is the delivery mechanism of choice
- Develop a novel set of interaction techniques for manipulating 3D objects, viewpoints, and animations.

8.1 Developing predictive models based on representative tasks

The current state of the art for creating end user applications is to build for a specific delivery mechanism. This is expensive and high risk. Our approach, based on a fundamental understanding of the human perceptual system, is to determine generic, representative tasks (such as object search, object selection, viewpoint manipulation, locomotion) and develop a predictive model. This predictive model allowed us to determine for a given end user application whether or not that application has attributes that merit the expense and difficulty of using a more exotic display mechanism. Developing this model involved a number of user studies comparing the effectiveness of HMDs, CAVEs, large screen displays, and traditional desktop displays for these representative tasks.

The development of interaction techniques is essentially a creative process. Our approach was to assemble a multidisciplinary team (including art majors, architecture majors, film studies majors, computer scientists, and design majors) and present them with interaction challenges such as: How would you build a virtual sand table once released from the constraints of the real world (such as gravity)?

8.2 Creating interactive programs

The Alice software system allows users to create interactive programs using 3D graphics, running on top of Windows platforms. Alice uses Direct 3D as its retained mode drawing package, and the Python programming language (www.python.org) as its scripting layer. The Alice Beta release included full support for implicit animation threads, the ability to import files in the .dxf and .obj format, and a revamped user interface.

9. Foreign-language Learning

This research is applying automated speech recognition to the acquisition and sustainment of foreign language speaking skills, which are increasingly important in an era of military and civilian multinational endeavors. Language skills are among the most expensive taught in the military, and atrophy quickly when not used.

Effective improvement of speaking skills requires a tutor to listen patiently to the student for extended periods of time and provide individualized feedback. Human tutors are costly and may be unavailable at the time or place where they are needed, such as military bases where soldiers need to keep up their skills in languages not spoken there. Previous language learning software either has not listened to the student, or has been limited in the feedback it provided, typically rejecting spoken utterances without being able to explain what was wrong with them. Our work seeks to overcome these limitations.

Key R & D issues include:

- How to elicit predictable speech
- How to model and detect disfluency, mispronunciation, and inappropriate prosody
- How to respond to oral language errors in a pedagogically effective manner
- How to make the user interface easy and engaging to use
- How to enable teachers to conveniently author materials for particular students

9.1 Approach

Our approach leverages from, and extends, Project LISTEN's past and present work on an automated reading coach that listens to children read aloud, and helps them when needed. This approach focuses on methods we can implement in the coach now so as to study their effectiveness in the context of real-time interactions with students. We adapt the reading coach into a pronunciation tutor for Spanish, as follows:

- To elicit predictable speech, we display text for the student to read aloud. This text represents an authentic task for Special Operations Forces — practicing the oral presentation of a speech that must be given fluently, intelligibly, and credibly, such as a "Mod-Demo" briefing in which SOF personnel introduce themselves to foreign diplomats. Value added includes the ability to tailor learning materials to actual mission-specific needs.
- To detect specific mispronunciations, we add them to the pronunciation dictionary used by the speech recognizer. These entries are generated automatically by phonological rules that represent transformations from native to accented speech. These rules were developed by Computational Linguistics graduate student Jeffrey Hill for his Master's Project at Carnegie Mellon. The choice of which types of mispronunciations to check for is based on advice from Spanish instructors about which types of errors their students commonly make, for example, "gringo-ized" versions of various Spanish vowels. Value added includes the ability to localize speaking errors to individual phonemes.
- To respond to phoneme-level mispronunciations, we present spoken feedback on the nature of the mistake (e.g., mispronouncing a vowel as a diphthong) and how to correct it (pronounce the pure vowel sound). Value added includes the ability to diagnose and remediate mistakes of particular types.

- To make the interface usable and effective, we put the learner in control of such choices as what to speak next, who (student or coach) is to speak it, and when to accept detailed tutorial feedback. We streamline the interaction to support the most frequent scenarios of usage. Value added includes the ability to tailor the interaction to student needs.

In parallel with our work on methods we can study "on-line" in the context of the current coach, we are performing "off-line" experiments to develop and evaluate new methods for analyzing speech prosody as well as pronunciation. This second approach provides the theoretical proof that the Sphinx II recognition system can be used to detect when a foreign speaker does not attain the intended articulatory target. A database of native and nonnative (ten different L1s) utterances of English is used. The nonnative speaker's utterance is compared to the natives' utterances.

9.2 Accomplishments

The funded work was presented in a video to the CAETI Community Conference, November, 1996.

9.3 Bibliography

[Eskenazi 96]

Eskenazi, M.

Detection of foreign speakers' pronunciation errors for second language training - preliminary results.

In *Proceedings of the Fourth International Conference on Spoken Language Processing*. October, 1996.

Philadelphia, PA.

[Mostow 96]

Mostow, J.

A Reading Tutor that Listens.

Video presented at the DARPA CAETI Community Conference, Berkeley, CA, Nov 1996.

Length: Five minutes.

[Mostow, Hauptmann, and Roth 95]

Mostow, J., A. Hauptmann, and S. Roth.

Demonstration of a Reading Coach that Listens.

In *Proceedings of the Eighth Annual Symposium on User Interface Software and Technology (UIST 95)*, pages 77-78. ACM SIGGRAPH and SIGCHI in cooperation with SIGSOFT, November, 1995.

Pittsburgh, PA.

Project LISTEN stands for "Literacy Innovation that Speech Technology ENables." We will demonstrate a prototype automated reading coach that displays text on a screen, listens to a child read it aloud, and helps where needed. We have tested successive prototypes of the coach on several dozen second graders. [Mostow et al AAAI94] reports implementation details and evaluation results. Here we summarize its functionality, the issues it raises in human-computer interaction, and how it addresses them. We are redesigning the coach based on our experience, and will demonstrate its successor at UIST '95.

10. Interactive Communication Technology

Our goal has been to create task-oriented technologies for collaboration among individuals, workgroups, and information services. Such collaborations will be mediated by computers to enhance access to relevant information or to overcome barriers of time, space, or culture. The key to such collaborations is communication in the context of information relevant to the task at hand. This means that collaboration must occur in the context of information tailored to the task at hand. Collaboration must also be able to cross language barriers, and all interactive software must support collaboration with relevant parties. Collaboration must not be confined to specialized software ghettos and must make effective use of all human communication modalities that can be applied to the task at hand (e.g., speech, gesture, handwriting, face-, eye-, pose-, body-motion). Finally, collaboration technologies must adapt to the tasks and the people involved, rather than the other way around.

The ICIE project has built on the premise that interactive information should be available anywhere, anytime, and to anyone who has authorization. The goal of our research was to extend the reach of desktop-interactive information into multiuser interaction and remote access.

10.1 Facilitators for collaboration

Pervasive collaboration

For successful computer-based collaboration it must be possible to communicate computer-based information to anyone at any time. This objective implies that collaboration cannot be confined to specialized applications; it must be broadly available in all applications. Our approach is to integrate collaboration deeply into the interactive software architecture. By defining an interactive surface between the application and its input/output devices, we can capture and distribute collaborative information in an application-independent manner. By managing and tracking change at the surface, we can make all participants aware of the work of their collaborators.

Multimodal web agents

The Carnegie Mellon Interactive Systems Lab has developed a new set of multimodal Web agents that provide more effective means to access, manipulate, generate, and disseminate multimedia information. By interpreting multimodal input, including speech, gesture, handwriting, and eye- and face- tracking this approach will achieve more rapid, natural, and effective interaction with multimedia material.

In Multimedia documents included video, speech, text, are accessed by spoken, gestured, and handwritten comments and commands. Suitable clips or segments of images, comments, and sounds, are then selected and manipulated by spontaneously spoken, gestured, or handwritten remarks to create new and value-added multimedia reports. Voice, drawings, gestures and handwritten notes can also be attached to the emerging document as multimodal annotations. A resulting multimodal Web document can then be shared with and disseminated to others, anywhere on the Web or on an Intranet, for further discussion, modification, and/or reporting. The multimedia documents (including the multimedia clips, the spoken and handwritten notes and annotations) are content searchable by speech, gesture, and character recognition technology.

10.2 Accomplishments

- We developed an information delta algebra for tracking changes in interactive behavior. This forms a foundation for collaborative interaction.
- We developed an architecture for multiagent participation in interactions. This design includes tools and techniques for drawing users' attention to information located by collaborators or automated agents working on the user's behalf.
- We developed a robust architecture for asynchronous collaboration. The information from any graphical application can be extracted and shared across the internet, without requiring collaborations to have identical software for all collaborative applications. Generalized tools were developed for coordinating changes generated by multiple collaborators on any given work product.
- We developed a generalized architecture for integrating observational agents with any application. Such agents intelligently analyze the graphical output of an application to perform their respective tasks. The agents are tools developed independently from any given application and then attached to the application to provide assistance.
- We developed tools to retarget graphical applications for speech-based information access. Spoken language access to graphical user interfaces provides remote access to desktop applications in contexts where screen, keyboard, or internet connection is not available. Semantic relationships are extracted from the graphical application and used as the basis for generating a spoken interpretation and supporting spoken-language navigation.

10.3 Bibliography

[Pausch, Proffitt, and Williams 97]

Randy Pausch, Dennis Proffitt, George Williams.

Quantifying Immersion in Virtual Reality.

In *Proceedings of SIGGRAPH 97*. ACM, July, 1997.

<http://www.cs.cmu.edu/~stage3/>

[publications/97/conferences/siggraph/immersion/](http://www.cs.cmu.edu/~stage3/publications/97/conferences/siggraph/immersion/).

Virtual Reality (VR) has generated much excitement but little formal proof that it is useful. Because VR interfaces are difficult and expensive to build, the computer graphics community needs to be able to predict which applications will benefit from VR. In this paper, we show that users with a VR interface complete a search task faster than users with a stationary monitor and a hand-based input device. We placed users in the center of the virtual room shown in Figure 1 and told them to look for camouflaged targets. VR users did not do significantly better than desktop users. However, when asked to search the room and conclude if a target existed, VR users were substantially better at determining when they had searched the entire room. Desktop users took 41% more time, re-examining areas they had already searched. We also found a positive transfer of training from VR to stationary displays and a negative transfer of training from stationary displays to VR.

11. Multimodal Interaction Technology

11.1 Multimodal error-repair

This research investigated cross-modal repair of errors in automatically recognized speech. Our work addressed methods that allow the user to switch modalities for corrections, e.g. from speech to handwriting, (oral) spelling, or gestures drawn on a touch-sensitive screen. We implemented cross-modal repair for dictation and form-filling applications and integrated these methods with the multimodal interaction mechanisms in our previously developed QuickTurn system.

User studies confirmed our hypothesis that cross-modal repair significantly expedites correcting speech-recognition errors, compared to respeaking and choosing from alternatives. We showed that cross-modal repair enables efficient repair without keyboard input.

These studies allowed us to evaluate repair extensively in the context of dictation applications. Keyboard input is efficient for text input and proficient users. However, productivity can be increased even for proficient users by using an automatic dictation system when the speech recognizer is highly accurate (>95%) and repair is efficient. In dictation applications efficient repair is possible both with cross-modal repair and using keyboard input.

We also investigated factors that drive user preference when they are given a choice among different repair modalities. Our studies clearly indicated that, while users tend to prefer speech initially, they learn to select the most efficient modality and to avoid those that prove less so.

11.2 A multimodal toolkit

We have developed a common infrastructure and design methodology for constructing multimodal user-interfaces. These ideas are embodied in an application framework and design toolkit.

Architecture for a multimodal-application framework

In the Multimodal Application Framework (MMApp) we have implemented a system architecture and a collection of software components that constitute it.

MMApp is modular, distributed, and customizable. The framework comprises reusable components or modules that export services via well-defined interfaces and hide implementation details. The framework's major components can run as separate processes distributed across multiple machines. MMApp provides reasonable defaults that are immediately useful in multimodal application development, but application developers always have the option of overriding the defaults and supplying alternative implementations to suit the needs of a particular application.

MMApp contains interfaces and implementations of speech/pen recording and recognition components, communication components used in distributed interprocess communication, and graphical user-interface components, including a default user interface in the form of a Java applet that can be deployed over the Web.

Multimodal design and rapid prototyping

The software components in MMAP can be instantiated and interconnected to create a multimodal application instance. To complete constructing the application, a multimodal interpreter must be instantiated for the target domain so that the application can determine the correct action to perform in response to an input event. We have developed a design process that can be followed to create a working multimodal application within the MMAP framework and the Multimodal Toolkit (MMTk), a collection of tools that automate many steps in the proposed design process.

MMTk contains a Visual Grammar Designer that allows constructing MMGL input models and modifying them using a drag-and-drop visual-construction paradigm. Other MMTk tools include a Random Sample Generator, an *N*-gram Language Model Generator, an Input Preprocessor Generator, an Integration Network Generator, and a Postprocessor Generator.

11.3 A multimodal workstation

We developed QuickTarget, a multimodal Web-based workstation for image analysts. By recognizing a combination of spontaneously-spoken utterances, gestures, and pointing, QuickTarget can process both images and video, allowing an analyst to describe and classify possible targets rapidly. QuickTarget also attaches speech, gesture, and handwritten annotations, recognizes spoken dictation for textual remarks, and, finally, generates multimedia Web-reports. We demonstrated QuickTarget over the Web, using a Web-based, multimodal server.

11.4 Communicator: A telephone-based dialog system

The Carnegie Mellon Communicator system is a telephone-based dialog system that supports travel planning, allowing users to create a travel itinerary based on actual availability. The Communicator system has served as a test bed for research in dialog management, natural language generation and speech recognition for real-time applications.

Based on our experiences with an earlier dialog management system, Script, we implemented a second dialog manager, Amoeba. Script supported linear dialogs, based on a form chain (that is, the task was broken down into a sequence of topics, each topic being handled by a single form, but the sequence of forms being pre-determined. Interaction followed this "script", though users were provided with navigational commands that allowed them to traverse the form chain). Amoeba generalized the Script model, adding facilities for defining abstract data structures and adding more powerful programming constructs. For recognition, we created a new set of acoustic models, incorporating actual Communicator data collected to that time using our system (a total of 2.5h). We created models from telephoned ATIS data (i.e., 25h of wideband (16kHz) speech, filtered to telephone bandwidth (8kHz)).

We increased the capacity of our system to handle 186 different U.S. destinations. In addition, we obtained a toll-free number for our system and began to publicize it externally (with the goal of generating use by non-developers).

We trained a new set of acoustic models for Communicator, based on 72 hours of Broadcast news, 10 hours of ATIS, and adapted using 7.1 hours of CMU Communicator speech. These models produced a word error rate of 27.4%. We incorporated a language-model backoff pattern

based confidence rating for decodings, keyed to individual words. We introduced a class-based language model for the Communicator, incorporating 19 classes.

We later began implementation of a new dialog manager, Agenda, which used a different approach from the scripting languages we had developed for Script and Amoeba. Agenda eliminates the form chain and replaces it with an ordered topic list derived from the itinerary data structure (itself now a dynamic data structure constructed over the course of an interaction). We completed a preliminary implementation.

We studied the feasibility of using a stochastic process for governing anaphora and like phenomena in generation and compared it to a heuristic approach. The stochastic system used a training corpus derived from transcriptions of a professional travel agent interacting with clients and modeled the likelihood of a given concept appearing in an utterance, given its occurrence in the immediately preceding turn. We found that while this technique produced acceptable output, a simple heuristic based on whether information was new or old (and whether it had been implicitly confirmed to the user) was sufficient.

CMU participated in the Communicator kick-off meeting in San Diego (13 January 1999), at which we gave a live demonstration of the Communicator.

11.5 Bibliography

[Bett et al. 00]

Bett, M., R. Gross, H. Yu, X. Zhu, Y. Pan, J. Yang, and A. Waibel.
Multimodal Meeting Tracker.
In *Proceedings of RIAO2000*. April, 2000.
Paris, France. To appear.

Face-to-face meetings usually encompass several modalities including speech, gesture, handwriting, and person identification. Recognition and integration of each of these modalities is important to create an accurate record of a meeting. However, each of these modalities presents recognition difficulties. Speech recognition must be speaker and domain independent, have low word error rates, and be close to real time to be useful. Gesture and handwriting recognition must be writer independent and support a wide variety of writing styles. Person identification has difficulty with segmentation in a crowded room. Furthermore, in order to produce the record automatically, we have to solve the assignment problem (who is saying what), which involves people identification and speech recognition. We follow a multimodal approach for people identification to increase the robustness (with the modules: color appearance id, face id and speaker id). This paper will examine a meeting room system under development at Carnegie Mellon University that enables us to track, capture and integrate the important aspects of a meeting from people identification to meeting transcription. Once a multimedia meeting record is created, it can be archived for later retrieval. This paper will review our meeting browser that we have developed which facilitates tracking and reviewing meetings.

[Jing et al. 97]

Jing, X., J. Yang, M.T. Vo, and A.H. Waibel.
Java front-end for Web-based multimodal human-computer interaction.
In *Proceedings of the Workshop on Perceptual User Interfaces (PUI97)*. November, 1997.
Banff, Alberta, Canada.

As the Java programming environment offers immediate capabilities for creating cross platform interactive applications, a Web-based, universal, multimodal interface becomes possible. In this paper we present a Java front-end for multimodal human/computer interaction. We address the problems in both system design and implementation. We demonstrate the feasibility of the proposed approach by a Web-based directory-assistant system and a multimodal interface for medical applications.

[Stiefelhagen and Yang 97]

Stiefelhagen, R. and J. Yang.
Gaze tracking for multimodal human-computer interaction.
In *Proceedings of ICASSP 97*. April, 1997.
Munich.

This paper discuss the problem of gaze tacking and its applications to multimodal human-computer interaction. The function of a gaze tracking system can be either passive or active. For example, a system can identify user's message target by monitoring the user's gaze, or launch an action by user's gaze. We have developed a real-time gaze tracking system that estimates the 3-D position and rotation (pose) of a user's head. We demonstrate the applications of the gaze tracker to human-computer interaction by two examples. The first example shows that gaze tracker can help speech recognition systems by switching language model and grammar based on user's gaze information. The second example illustrates the combination of the gaze tracker and a speech recognizer to view a panoramic image.

[Stiefelhagen, Yang, and Waibel 97]

Stiefelhagen, R., J. Yang, and A.H. Waibel.

A model-based gaze-tracking system.

*Int. Journal of Artificial Intelligence Tools*6(2):193-209, 1997.

In this paper we present a nonintrusive, model-based, gaze-tracking system. The system estimates the 3-D pose of a user's head by tracking as few as six facial feature points. The system locates a human face using a statistical color model and then finds and tracks the facial features, such as eyes, nostrils and lip corners. A full perspective model is employed to map these feature points onto the 3-D pose. Several techniques have been developed to track the features points and recover from failure. We currently achieve a frame rate of 15+ frames per second using an HP 9000 workstation with a framegrabber and a Canon VC-C1 camera. The application of the system has been demonstrated by a gaze-driven, panoramic image-viewer. Potential applications of the system include multimodal interfaces, virtual reality, and video-teleconferencing.

Keywords: Gaze tracking, head orientation, human-computer interaction, multimodal interfaces, facial feature extraction

[Suhm 97]

Suhm, B.

Empirical evaluation of interactive multimodal error recovery.

In *Proceedings of the IEEE Workshop on Speech Recognition and Understanding (ASRU)*.

IEEE, December, 1997.

Santa Barbara, CA.

Commercial dictation systems for continuous speech have recently become available. Although they generally received positive reviews, error correction is still limited to choosing from a list of alternatives, speaking, or typing. We developed a set of interactive methods to correct errors without using either keyboard or mouse, thus allowing the user to switch among continuous-speech, spelling, handwriting, and pen-gesture modalities. These correction methods were integrated with our large-vocabulary speech-recognition system to build a prototype, multimodal, listening typewriter. The efficiency of different error correction methods was evaluated in a user study. The experiment compares multimodal correction with other methods available in current speech-recognition applications. Results confirm the hypothesis that switching among modalities can significantly expedite corrections. Thus, state-of-the-art speech-recognition technology with multimodal error correction makes it possible to enter text faster than unaided typing, including the time necessary to correct errors. In applications where a keyboard is acceptable, however, typing still remains the fastest method to correct errors for users with good typing skills.

[Suhm 98]

Suhm, B.

Multimodal Interactive Error Recovery for Non-Conversational Speech User Interfaces.

PhD thesis, Universitat Karlsruhe, July, 1998.

Available via <http://werner.ira.uka.de/~bsuhm/thesis/thesis.ps.gz>.

This dissertation presents an interactive, multimodal approach for efficient, keyboard-free, error correction in nonconversational speech-recognition applications that employ graphic user interfaces. This approach improves efficiency of keyboard-free correction in two ways: First, by switching input modalities for correction and, second, by correlating correction input with the context of a repair. By correlating correction input with repair context, correction accuracies (success rate) of 80-90% are achieved despite the difficulty of recognizing correction input.

To evaluate empirically the efficiency of multimodal correction in a potentially useful

application, a prototype multimodal text editor was built, integrating interactive multimodal correction with a state-of-the-art, large-vocabulary continuous speech recognizer. User studies then compared multimodal methods with conventional keyboard- and mouse-based correction methods on a dictation task. Results show that without use of a keyboard, text input rates of more than 40 wpm are feasible, assuming 90% accurate recognition of dictation input in real time. This rate compares favorably to fast, non-secretarial typing. This research thus confirms the hypothesis that multimodal flexibility can accelerate error correction in speech recognition applications. Our study shows that user preferences of correction modality are driven by the level of correction accuracy. It confirms the common assumption that most users would prefer speech for text-input tasks if speech correction were as accurate as other modalities. Multimodal input is more efficient than keyboard for users with poor typing skills, for users suffering from Carpal Tunnel Syndrome, and in applications where keyboard input is slow, such as with small handheld devices.

[Suhm and Waibel 97]

Suhm, B. and A.H. Waibel.

Exploiting repair context in interactive error recovery.

In *Proceedings of Eurospeech 97*. September, 1997.

Rhodes, Greece.

In current speech applications, facilities to correct recognition errors are limited to either choosing among alternative hypotheses (either by voice or by mouseclick) or respeaking. Information from the context a repair is ignored. We developed a method that improves the accuracy of correcting speech recognition errors interactively by taking into account the context of the repair interaction. The basic idea is to use the same language modeling information used in the initial decoding of continuous speech input for decoding (isolated word) repair input. The repair is not limited to speech, but the user can choose to switch modality, for instance spelling or hand-writing a word. We implemented this idea by rescoring N-best lists obtained from decoding the repair input using language-model scores for trigrams that include the corrected word. We evaluated the method on a set of repairs by respeaking, spelling, and writing, and collected data with our prototype, continuous-speech, dictation interface. The method can increase the accuracy of repair significantly, compared to recognizing the repair input as independent event.

[Tebelskis 95]

Tebelskis, J.

Speech Recognition using Neural Networks.

PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1995.

Available as technical report CMU-CS-95-142.

This thesis examines how artificial neural networks can benefit a large vocabulary, speaker independent, continuous speech recognition system. Currently, most speech recognition systems are based on hidden Markov models (HMMs), a statistical framework that supports both acoustic and temporal modeling. Despite their state-of-the-art performance, HMMs make a number of suboptimal modeling assumptions that limit their potential effectiveness. Neural networks avoid many of these assumptions, while they can also learn complex functions, generalize effectively, tolerate noise, and support parallelism. While neural networks can readily be applied to acoustic modeling, it is not yet clear how they can be used for temporal modeling. Therefore, we explore a class of systems called NN-HMM hybrids, in which neural networks perform acoustic modeling, and HMMs perform temporal modeling. We argue that a NN-HMM hybrid has several theoretical advantages over a pure HMM system, including better acoustic modeling accuracy, better context sensitivity, more natural discrimination, and a more economical use of parameters. These ad-

vantages are confirmed experimentally by a NN-HMM hybrid that we developed, based on context-independent phoneme models, that achieved 90.5% word accuracy on the Resource Management database, in contrast to only 86.0% accuracy achieved by a pure HMM under similar conditions.

[Vo 98]

Vo, M.T.

A Framework and Toolkit for the Construction of Multimodal Learning Interfaces.

PhD thesis, School of Computer Science, Carnegie Mellon University, April, 1998.

Available as technical report CMU-CS-98-129 .

Multimodal human-computer interaction, in which the computer accepts input from multiple channels or modalities, is more flexible, natural, and powerful than unimodal interaction with input from a single modality. Many research studies have reported that the combination of human communication means such as speech, gestures, handwriting, eye movement, etc., enjoys strong preference among users. Unfortunately, the development of multimodal applications is difficult and still suffers from a lack of generality, such that a lot of duplicated effort is wasted when implementing different applications sharing some common aspects. The research presented in this dissertation aims to provide a partial solution to the difficult problem of developing multimodal applications by creating a modular, distributed, and customizable infrastructure to facilitate the construction of such applications.

This dissertation contributes in three main areas: theory of multimodal interaction, software architecture and reusable application framework, and rapid application prototyping by domain-specific instantiation of a common underlying architecture.

The foundation of the application framework and the rapid prototyping tools is a model of multimodal interpretation based on semantic integration of information streams. This model supports most of the conceivable human communication modalities in the context of a broad class of applications, specifically those that support state manipulation via parameterized actions. The multimodal semantic model is also the basis for a flexible, domain-independent, incrementally trainable multimodal interpretation algorithm based on a connectionist network.

The application framework and design process have been successfully applied to the construction of three multimodal systems in three different domains.

[Waibel et al. 97]

Waibel, A.H, B. Suhm, M.T. Vo, and J. Yang.

Multimodal interfaces for multimedia information agents.

In *Proceedings of ICASSP 97*. April, 1997.

Munich.

When humans communicate, they take advantage of a rich spectrum of cues. Some are verbal and acoustic. Some are nonverbal and nonacoustic. Signal processing technology has devoted much attention to the recognition of speech as a single human communication signal. Most other complementary communication cues, however, remain unexplored and unused in human-computer interaction. In this paper we show that the addition of nonacoustic or nonverbal cues can significantly enhance robustness, flexibility, naturalness and performance of human-computer interaction. We demonstrate computer agents that use speech, gesture, handwriting, pointing, spelling jointly for more robust, natural and flexible human-computer interaction in the various tasks of an information worker: information creation, access, manipulation or dissemination.

[Yang et al. 98]

Yang, J., R. Stiefelhagen, U. Meier, and A.H. Waibel.

Visual tracking for multimodal human computer interaction.

In C. Karat, A. Lund, J. Coutaz, and J. Karat (editors), *Proceedings of the CHI '98 Conference on Human Factors in Computing Systems*, pages 140-147. ACM, April, 1998. Los Angeles.

In this paper we present visual tracking techniques for multimodal human/computer interaction. First, we discuss techniques for tracking human faces, techniques that use human skin-color as a major feature. An adaptive stochastic model has been developed to characterize skin-color distributions. Based on the maximum likelihood method, the model parameters can be adapted for different people and different lighting conditions. The feasibility of the model has been demonstrated by the development of a real-time face tracker. The system has achieved a rate of 30+ frames/second using a low-end workstation with a framegrabber and a camera. We also present a top-down approach for tracking facial features such as eyes, nostrils, and lip corners. These realtime visual tracking techniques have been successfully applied to many applications, such as gaze-tracking and lip-reading. The face tracker has been combined with a microphone array for extracting a speech signal from a specific person. The gaze tracker has been combined with a speech recognizer in a multimodal interface for controlling a panoramic image viewer.

[Yang et al. 00]

Yang, J., X. Zhu, R. Gross, J. Kominek, Y. Pan, and A. Waibel.

Multimodal People ID for a Multimedia Meeting Browser.

In . 2000.

To appear.

A meeting browser is a system that allows users to review a multimedia meeting record from a variety of indexing methods. Identification of meeting participants is essential for creating such a multimedia meeting record. Moreover, knowing who is speaking can enhance the performance of speech recognition and indexing meeting transcription. In this paper, we present an approach that identifies meeting participants by fusing multimodal inputs. We use face ID, speaker ID, color appearance ID, and sound source directional ID to identify and track meeting. After describing the different modules in detail, we will discuss a framework for combining the information sources. Integration of the multimodal people ID into the multimedia meeting browser is in its preliminary stage.

[Yang, Lu, and Waibel 98]

Yang, J., W. Lu, and A.H. Waibel.

Skin-color modeling and adaptation.

In *Proceedings of the Asian Conference on Computer Vision (ACCV'98)*, Vol II, pages 687-694. January, 1998.

Hong Kong.

This paper studies a statistical skin-color model and its adaptation. It is revealed that (1) human skin colors cluster in a small region in a color space; (2) the variance of a skin color cluster can be reduced by intensity normalization, and (3) under a certain lighting condition, a skin-color distribution can be characterized by a multivariate normal distribution in the normalized color space. We then propose an adaptive model to characterize human skin-color distributions for tracking human faces under different lighting conditions. The parameters of the model are adapted based on the maximum likelihood criterion. The model has been successfully applied to a real-time face tracker and other applications.

12. Rapidly Deployable Speech Translation

DIPLOMAT is a pilot project in rapid-deployment, unrestricted speech-to-speech translation. The system will be retargetable to both new languages and new domains orders of magnitude more quickly than commercial technology. Unrestricted coverage will be achieved through user-driven incremental improvement. Initial test languages are Serbo-Croatian, Korean, Haitian-Creole, and Arabic (all to/from English).

12.1 Towards bi-direction language translation

We have developed a rapidly deployable, bidirectional, wearable, speech-to-speech translation system. The DIPLOMAT system offers significant speech-to-speech translation capabilities between English and four other languages: Croatian, Spanish, Haitian Creole, and Korean. Our efforts have been primarily targeted at rapid deployment of new languages on a wearable platform, for use by peace-keeping forces for such tasks as force protection interviews. DIPLOMAT is retargetable to both new languages and new domains orders of magnitude faster than commercial MT technology allows.

We have achieved wide coverage and error-compensation through user-interaction. The system is intended for use on wearable computers in authentic field conditions. The initial demonstration involved creating an initial bidirectional Serbo-Croatian/English-speech MT system in several weeks and improving coverage over several months. DIPLOMAT combines and extends previous DARPA- and DoD-funded research at Carnegie Mellon University in machine translation, continuous-speech recognition, speech synthesis, and wearable computers.

This bidirectional MT system is significantly more challenging than the parallel "one-way" DARPA-funded effort, having as its objective the ability to translate general responses from naive, untrained foreign-language speakers. This is very challenging from both MT and user-interface points of view. For MT, the less restricted the topics of discussion, the more difficult the translation problem. For user interfaces, we must be able to interact effectively with someone who may never have seen a computer before. DoD reaction to our initial demonstration made clear the need to focus additional effort on developing simple yet powerful graphical user interfaces (GUIs), to allow naive foreign-language speakers to interactively correct speech recognition and translation errors with no training.

Concurrently with producing demonstration systems in these languages, we also carried out research into improving the underlying technology for rapid-deployment MT, in particular rapid-deployment morphological analysis (RDMA) and generalized Example-Based MT (EBMT). RDMA is currently the development-time bottleneck for translating new languages. One can finesse RDMA for languages such as Serbo-Croatian and Haitian Creole, but it will be absolutely crucial for a usable Arabic system. EBMT uses corresponding pairs of sentences in the two languages as examples for translating new sentences that contain sub-phrases of the example sentences. EBMT is a primary contributor to the rapid-deployment MT capabilities of DIPLOMAT.

12.2 Applying DIPLOMAT: Building a translating telephone

As a three-month extension to the DIPLOMAT project, we proposed to produce a Translating Telephone, allowing any two individuals with telephones and web browsers to communicate across a language barrier, with no advance installation required. We proposed to initially develop this system for bidirectional Korean/English speech-to-speech translation, with a potential application facilitating cooperation between US and South Korean forces.

The primary purpose of the web browser is not long-distance communication across the Web (although that is an important side-benefit), but rather that two people even in the same room can bring up the Translating Telephone without any special hardware or advance software installation. The proposed system would be designed using "plug-ins" to a standard web browser such as Netscape, plus an audio connection that could be provided by an ordinary voice telephone.

Thus, in the case of Korea, if there is trouble communicating between cooperating forces, they can start up the translator just by bringing up Netscape and downloading our plug-in, wherever they are. They do not need to have previously installed it or have any special hardware. It will be immediately available to all our forces, anywhere that they have a Web connection and a telephone.

12.3 Implementing the translating telephone

We wrote the user interface of the translating telephone in Java, and interfaced to a Central Server, which coordinated communications between components. This architecture was required due to the security constraints that Java enforces. The architecture of the system is described in (Hogan and Frederking, 2000).

In addition, we produced an optional Editor client to allow a human language expert to be present during initial development phase of a new language, while the system is still learning.

We implemented a web interface to allow selection of language pair, and send email to the other interlocutor advising him/her how to use the system.

We demonstrated the system using only close-talking microphones. No demonstration was given using a telephone as speech input device.

We demonstrated full speech-to-speech capabilities in Spanish/English in April 1999. Following the demonstration, we improved the ergonomics of the interface according to suggestions of attendees. No demonstration of the complete system was given for Korean/English, although all components of the system have been shown to work for that language pair.

12.4 Accomplishments in language translation

- Developed a much improved Korean/English version of the text-text and speech-speech translator, demonstrating rapid deployment for an Asian language. This wearable system is suitable for initial field testing by Service personnel in realistic scenarios, such as:
 - Korean speech data collection

- Korean speech synthesis (SS)
- Adaptation of MT system to Korean (16-bit char)
- GUI adaptation for Korean characters
- Collected a major database of Korean speech data. As far as we know, this is the largest existing Korean speech database: 4769 Mbyte, 274 speakers.
- Developed a Haitian-Creole/English version of the speech translator, demonstrating rapid deployment for a different language type
- Collected a major database of Haitian-Creole speech data: 2979 Mbyte, 149 speakers
- Developed a Spanish/English version of the speech translator, assembled from available components (except for speech synthesis) in order to allow interesting non-DoD demonstrations.
- Developed techniques for translating between languages with highly-divergent syntax (Korean and English, for example). Tagging and bracketing techniques allow our rapid-deployment approach to produce reasonably good translations, even where surface word-order differs significantly.
- Developed the TEXT-R newswire text-selection program to reduce redundancy in and improve the quality of our parallel corpus-development effort
- Demonstrated experimentally using our Croatian/English system that our MEMT translation architecture is effective
- Integrated the Haitian-Creole/English translation component into the Army Research Lab's FALCON system for field prioritization of written material in Haiti. Produced an OCR-error corrector for FALCON, similar in concept to a spelling corrector but incorporating knowledge of OCR errors and a statistical language model.
- Ported the Serbo-Croatian system from Windows laptops to the SMART Module of the Carnegie Mellon/DARPA Language Translator.

12.5 Related accomplishments

- Architecture evaluation: The first-ever evaluation of the Multi-Engine Machine Translation (MEMT) architecture was published as (Hogan & Frederking, 1998). This evaluation demonstrates that using many translation engines and combining them is indeed better than any single engine.
- Further MT improvements: Improvements to the Example-Based MT system are described in (Brown, 1999). One of the key results is a substantial reduction in the amount of bilingual text needed to produce the same level of coverage.
- ARL Collaboration: Our on-going work with the Army Research Lab resulted in the deployment of our Haitian Creole translation system as part of the FALCON (Forward Area Language Converter). This collaboration led to work on Optical Character Recognition for Haitian Creole cf. (Hogan, 1998) and (Hogan, 1999).
- Haitian Creole Orthography: As part of an effort to standardize the written forms in our language database for Haitian Creole, we investigated the phonetics behind a particularly common alternation (Hogan & Allen, 1999).
- System Description: A complete description of the DIPLOMAT system, its goals

and methodology is being published as part of a special issue of the journal Machine Translation on Speech Translation Technology, .

12.6 Technology transition

In November 1998, we began investigations into possible joint work in English/French Speech-to-Speech Machine Translation with The University of Nancy in France, with the help of Raj Reddy and Jean-Paul Haton. The original DIPLOMAT software was ported to Nancy, and a the software was adapted to permit networked communication using two computers (the DIPLOMAT system was designed to run on one computer).

We are currently involved in a DIPLOMAT spin-off, the Audio Voice Translation Guide System (TONGUES). TONGUES is being investigated under contract to Lockheed-Martin (Oswego, NY), and is funded by the U.S. Army ACT II Program. The project has a December 2000 deadline to produce a prototype field version of the wearable DIPLOMAT system for use by the Army Chaplaincy and other units.

12.7 Bibliography

[Allen 98]

Allen, J.

Lexical variation in Haitian Creole and orthographic issues for MT and OCR applications. In *Proceedings of the Workshop on Embedded MT Systems*. Association for Machine Translation in the Americas, October, 1998.

Langhorne, PA. Held in conjunction with AMTA'98.

[Allen and Hogan 98a]

Allen, J., and C. Hogan.

Evaluating Haitian Creole orthographies from a non-literacy-based perspective.

In *Proceedings of the Annual Meeting of the Society for Pidgin and Creole Linguistics & Linguistics Society of America*. January, 1998.

New York City.

[Allen and Hogan 98b]

Allen, J., and C. Hogan.

Expanding lexical coverage of parallel corpora for the Example-Based Machine Translation approach.

In *Proceedings of the First International Conference on Language Resources and Evaluation*, pages 747-754. May, 1998.

Granada, Spain. Vol. 2.

In this paper we discuss a method called TEXTR that improves lexical coverage in creating parallel corpora that are to be subsequently implemented in an Example-Based Machine Translation (EBMT) system. First, we explain the purpose and importance of the EBMT approach. Second, we indicate how low-density languages can benefit from rapid corpora development using our method as compared to other corpora-expansion techniques. Third, an evaluation of these various methods, based on tests for current lexical coverage, projected lexical coverage, and word-frequency distribution, clearly indicates that TEXTR is a valid mechanism for EBMT corpus development. We conclude that this method will improve the quality and speed with which parallel corpora are created, both for our specific purposes and possibly for other corpus-based applications.

[Brown 96]

Brown, R.D.

Example-Based Machine Translation in the Pangloss System.

In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*. August, 1996.

Copenhagen, Denmark.

The Pangloss Example-Based Machine Translation engine (PanEBMT) is a translation system requiring essentially no knowledge of the structure of a language, merely a large, parallel corpus of example sentences and a bilingual dictionary. Input texts are segmented into sequences of words occurring in the corpus, for which translations are determined by subsentential alignment of the sentence pairs containing those sequences. These partial translations are then combined with the results of other translation engines to form the final translation produced by the Pangloss system. In an internal evaluation, PanEBMT achieved 70.2% coverage of *unrestricted* Spanish news-wire text, despite a simplistic, subsentential alignment algorithm, a suboptimal dictionary, and a corpus from a different domain than the evaluation texts.

[Brown 98]

Brown, R.D.

Improving Embedded MT with User Interaction.

In *Proceedings of the Workshop on Embedded MT Systems*. Association for Machine Translation in the Americas, October, 1998.

Langhorne, PA. Held in conjunction with AMTA'98.

Automated machine translation (MT) of unrestricted texts is known to be error-prone, and thus typically requires human involvement in the translation process to achieve acceptable quality. In a conventional system performing interactive speech translation between two speakers, neither pre-editing nor conventional post-editing is possible, and the users will generally have no direct access to the embedded MT component. Since most translation errors are the result of incorrect selections, one approach to solving the dilemma is to provide the user with access to various alternative translations and allow him to select the one which appears to be correct in context. If the system remembers the manual selection, it can make the correct selections automatically whenever the same context is encountered thereafter.

[Brown 99]

Brown, R.

Adding Linguistic Knowledge to a Lexical Example-Based Translation System.

In *Proceedings of the Eighth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-99)*, pages 22-32. August, 1999.

Chester, England.

Example-Based Machine Translation (EBMT) using partial exact matching against a database of translation examples has proven quite successful, but requires a large amount of pre-translated text in order to achieve broad coverage of unrestricted text. By adding linguistically tagged entries to the example base and permitting recursive matches that replace the matched text with the associated tag, substantial reductions in the required amount of pre-translated text can be achieved. A modest investment of time -- on the order of two person-weeks -- adding linguistic knowledge reduces the required example text by a factor of six or more, while retaining comparable translation quality. This reduction makes EBMT more attractive for so-called "low-density" languages for which little data is available.

[Brown and Frederking 95]

Brown, R.D. and R.E. Frederking.

Applying Statistical English Language Modelling to Symbolic Machine Translation.

In *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation (TMI'95)*, pages 221-239. July, 1995.

Leuven, Belgium.

The Pangloss Mark III system was, from the outset, designed to be a symbolic, human-aided machine-translation (MT) system. The need arose to rapidly adapt it for use as a fully-automated MT system. Our solution to this problem was to add a statistical English language model (ELM) to replace the most significant user activity, selecting between alternate translations produced by the system. The language model used is a trigram model with backoff to bigram and unigram probabilities. The language modeling and search procedure are described in detail, and comparison is made to other trigram-based statistical MT work.

[Eskenazi et al. 97]

Eskenazi, M., C. Hogan, J. Allen, and R.E. Frederking.
 Issues in database creation: recording new populations, faster and better labelling.
 In *Proceedings of Eurospeech 97*, pages 1699-1703. September, 1997.
 Rhodes, Greece. Vol. 4, poster session.

[Eskenazi et al. 98]

Eskenazi, M., C. Hogan, J. Allen, and R.E. Frederking.
 Issues in database design: recording and processing speech from new populations.
 In *Proceedings of the First International Conference on Language Resources and Evaluation*,
 pages 1289-1293. May, 1998.
 Granada, Spain. Vol. 2, poster session.

[Frederking et al. 99]

Frederking, R., Hogan, C., and Rudnicky, A.
 A New Approach to the Translating Telephone.
 In *Proceedings of the Machine Translation Summit VII: MT in the Great Translation Era*.
 September, 1999.
 Singapore.

The Translating Telephone has been a major goal of speech translation for many years. Previous approaches have attempted to work from limited-domain, fully-automatic translation towards broad-coverage, fully-automatic translation. We are approaching the problem from a different direction: starting with a broad-coverage but not fully-automatic system, and working towards full automation. We believe that working in this direction will provide us with better feedback, by observing users and collecting language data under realistic conditions, and thus may allow more rapid progress towards the same ultimate goal. Our initial approach relies on the wide-spread availability of Internet connections and web browsers to provide a user interface. We describe our initial work, which is an extension of the DIPLOMAT wearable speech translator project.

[Frederking et al. 00a]

Frederking, R., Rudnicky, A., Hogan, C., and Lenzo, K.
 Interactive Speech Translation in the DIPLOMAT Project. Machine Translation.
 In *Machine Translation on Speech Translation Technology*. 2000.
 To appear.

The DIPLOMAT rapid-deployment speech translation system is intended to allow naive users to communicate across a language barrier, without strong domain restrictions, despite the error-prone nature of current speech and translation technologies. In addition, it should be deployable for new languages an order of magnitude more quickly than traditional technologies. Achieving this ambitious set of goals depends in large part on allowing the users to interactively correct recognition and translation errors. We briefly present the Multi-Engine Machine Translation (MEMT) architecture, describing how it is well-suited for such an application. We then discuss our approaches to rapid-deployment speech recognition and synthesis. Finally we describe our incorporation of interactive error correction throughout the system design. We have already developed working bidirectional Croatian <-> English and Spanish <-> English systems, and have Haitian Creole <-> English and Korean <-> English versions under development.

[Frederking et al. 00b]

Hogan, C. and Frederking, R.

WebDIPLOMAT: A Web-Based Interactive Machine Translation System.

In *Proceedings of the 18th International Conference on Computational Linguistics (COLING '2000)*. August, 2000.

Saarbruecken, To appear.

We describe an interactive, web-based, chat-style machine translation system, supporting speech recognition and synthesis, local- or third-party correction of speech recognition and machine translation output and online learning. The underlying client-server architecture, implemented in Java, provides remote, distributed computation for the translation and speech subsystems. We further describe web-based user interfaces which are easily redesigned to produce a number of useful configurations.

[Frederking, Rudnicky, and Hogan 97]

Frederking, R.E., A.I. Rudnicky, and C. Hogan.

Interactive Speech Translation in the DIPLOMAT Project.

In *Proceedings of the Spoken Language Translation Workshop at (ACL 97)*, pages 61-66. Association for Computational Linguistics, May, 1997.

Madrid, Spain. An extended version has been submitted to the *Machine Translation Journal*.

The DIPLOMAT rapid-deployment speech translation system is intended to allow naive users to communicate across a language barrier, without strong domain restrictions, despite the error-prone nature of current speech and translation technologies. Achieving this ambitious goal depends in large part on allowing the users to correct recognition and translation errors interactively. We briefly present the Multi-Engine Machine Translation (MEMT) architecture, describing how it is well-suited for such an application. We then describe our incorporation of interactive error correction throughout the system design. We have already developed a working bidirectional Serbo-Croatian/English system, and are currently developing Haitian-Creole/English and Korean/English versions.

[Hogan 98]

Hogan, C.

Correcting OCR'd Text for Machine Translation.

In *Proceedings of the Workshop on Embedded MT Systems*. Association for Machine Translation in the Americas, October, 1998.

Langhorne, PA. Held in conjunction with AMTA'98.

While commercial Optical Character Recognition (OCR) systems are quite good, they often fail to incorporate any language-specific error correction component. In this paper we describe an OCR post-processor that incorporates language-specific knowledge using a Statistical Language Modeller (SLM). Because the post-processor is outside of the OCR software, the language it recognizes may be easily changed, enabling generic OCR software to be reliably used for many languages. We specifically address the problem of developing OCR for economically less viable languages, such as Haitian Creole.

[Hogan 99]

Hogan, C.

OCR for Minority Languages.

In *Proceedings of the 1999 Symposium on Document Image Understanding Technology (SDIUT '99)*, pages 235-244. April, 1999.

Annapolis, MD.

In this paper I discuss the difficulties encountered when applying Optical Character Recognition (OCR) to minority languages. In particular, I explore the case of developing

OCR for Haitian Creole (HC), a vernacular, minority language. Although HC is written with a variant of the Roman alphabet, no OCR device has ever been developed specifically with HC in mind, with the result that recognition can be fairly poor. I present a technique for post-processing OCR output that is independent of the OCR device being used, and demonstrate that it can improve OCR recognition for HC.

[Hogan and Allen 99]

Hogan, C. and Allen, J.

Phonemic and Orthographic Realizations of 'r' and 'w' in Haitian Creole.

In *Proceedings of the 14th International Congress of Phonetic Sciences (ICPHS '99)*. August, 1999.

San Francisco, CA.

This paper presents a synchronic perspective on the phonemic status of the orthographic forms 'r' and 'w' that appear in Haitian Creole (HC) texts. Other HC language researchers have postulated two phonemes (i.e., /r/ and /w/) conditioned by roundness/labialization. Such evidence is contradicted by written corpora. Our analyses take into account the variation in HC found in written and spoken corpora. From this work, we aim to determine the status and distribution of the related phonemes and phonetic realizations in HC. Our findings have considerable bearing on speech recognition and speech synthesis systems that are currently under development for HC and other languages.

[Hogan and Frederking 98]

Hogan, C. and R.E. Frederking.

An Evaluation of the Multi-Engine MT Architecture.

In L. Gerber and D. Farwell (editors), *Machine Translation and the Information Soup*:

Proceedings of the Third Conference of the Association for Machine Translation in the Americas (AMTA '98). October, 1998.

Langhorne, PA. Lecture Notes in Computer Science. (Berlin) Springer-Verlag.

The Multi-Engine MT (MEMT) architecture combines the outputs of multiple MT engines using a statistical language model of the target language. It has been used successfully in a number of MT research systems, for both text and speech translation. Despite its perceived benefits, there has never been a rigorous, published, double-blind evaluation of the claim that the combined output of a MEMT system is in fact better than that of any one of the component MT engines. We report here the results of such an evaluation. The combined MEMT output is shown to indeed be better overall than the output of the component engines in a Croatian/English MT system. This result is consistent in both translation directions, and between different raters.

[Lenzo, Hogan, and Allen 98]

Lenzo, K., C. Hogan, and J. Allen.

Rapid-Deployment Text-to-Speech in the DIPLOMAT System.

In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP98)*. November, 1998.

Sydney, Australia. Presented in poster session.

The DIPLOMAT project at Carnegie Mellon University instantiates a program of rapid-deployment speech-to-speech machine translation. We have developed techniques for quickly producing text-to-speech (TTS) systems for new target languages to support this work. While the resulting systems are not immediately of comparable quality to commercial systems on unrestricted tasks in well-developed languages, they are more than adequate for limited-domain scenarios and rapid prototyping — they generalize to unseen data with some degradation, while quality in-domain can be quite good. Voices and engines for syn-

thesizing new target languages may be developed in a period as short as two weeks after text corpus collection. We have successfully used these techniques to build a TTS module for English, Croatian, Spanish, Haitian Creole and Korean.